



PATENT

IN THE UNITED STATES PATENT & TRADEMARK OFFICE

Inventor:	KUSHWAH et al.	Examiner:	Belix M. Ortiz
Application No.:	10/816,202	Art Unit:	2164
Filed:	March 31, 2004	Docket No.:	LEGAP024
Title:	SELECTIVE DATA RESTORATION		

DECLARATION UNDER 37 CFR § 1.131

Mail Stop RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

We, Ajay Pratap Singh Kushwah & Venkatesha Murthy, declare as follows:

1. Exhibits A – C show a version control tool used to track bugs and changes in the source tree and in particular shows saved information associated with the identifier “LGTpa45351.” Exhibit A shows the problem description, Exhibit B shows the resolution description, and Exhibit C shows the source code files (including version number and date) associated with identifier “LGTpa45351.” The source code files associated with identifier “LGTpa45351” include an actual reduction to practice of the subject matter recited in claims 1, 20, and 21; versions prior to “LGTpa45351” do not include an actual reduction to practice of the subject matter recited in claims 1, 20, and 21.

2. Exhibit D shows the differences between source code file BigCacheInterfaces.cpp version number 1.5.16.3 with a date of October 19, 2002 and the version immediately prior. Exhibit E shows the differences between source code

Application Serial No. 10/816,202
Attorney Docket No. LEGAP024

v. venkatesha murthy

actual practice in the source code files associated with identifier "LGTpa45351" on or before October 29, 2002.

4. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Ajay Pratap Singh Kushwah

Date

V. Venkatesha Murthy.
Venkatesha Murthy

August 01, 2007
Date



PATENT

IN THE UNITED STATES PATENT & TRADEMARK OFFICE

Inventor:	KUSHWAH et al.	Examiner:	Belix M. Ortiz
Application No.:	10/816,202	Art Unit:	2164
Filed:	March 31, 2004	Docket No.:	LEGAP024
Title:	SELECTIVE DATA RESTORATION		

DECLARATION UNDER 37 CFR § 1.131

Mail Stop RCE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

We, Ajay Pratap Singh Kushwah & Venkatesha Murthy, declare as follows:

1. Exhibits A – C show a version control tool used to track bugs and changes in the source tree and in particular shows saved information associated with the identifier “LGTpa45351.” Exhibit A shows the problem description, Exhibit B shows the resolution description, and Exhibit C shows the source code files (including version number and date) associated with identifier “LGTpa45351.” The source code files associated with identifier “LGTpa45351” include an actual reduction to practice of the subject matter recited in claims 1, 20, and 21; versions prior to “LGTpa45351” do not include an actual reduction to practice of the subject matter recited in claims 1, 20, and 21.


2. Exhibit D shows the differences between source code file BigCacheInterfaces.cpp version number 1.5.16.3 with a date of October 19, 2002 and the version immediately prior. Exhibit E shows the differences between source code


file CelestraBigImpl.cpp version number 1.16.6.16 with a date of October 19, 2002 and the version immediately prior. Exhibit F shows the differences between source code file CelestraBigImpl.hpp version number 1.5.16.4 with a date of October 20, 2002 and the version immediately prior. Exhibit G shows the differences between source code file MDImage.cpp version number 1.1.2.2 with a date of October 19, 2002 and the version immediately prior. Exhibit H shows the differences between source code file MDImage.hpp version number 1.1.2.2 with a date of October 19, 2002 and the version immediately prior. Exhibit I shows the differences between source code file getNext.hpp version number 1.1.2.2 with a date of October 19, 2002 and the version immediately prior. Exhibit J shows the differences between source code file rip.cpp version number 1.1.2.15 with a date of October 19, 2002 and the version immediately prior. Exhibit K shows the differences between source code file rip.hpp version number 1.1.2.3 with a date of October 19, 2002 and the version immediately prior. Exhibit L shows the differences between source code file rtrv_filemd.cpp version number 1.1.2.11 with a date of October 19, 2002 and the version immediately prior. Exhibit M shows the differences between source code file rtrv_filemd.hpp version number 1.1.2.2 with a date of October 19, 2002 and the version immediately prior. Exhibit N shows the differences between source code file rtrvsinglepass.cpp version number 1.1.2.12 with a date of October 29, 2002 and the version immediately prior.

3. Identifying a file system element for restoration by "receiving a request to restore a file system element; determining an offset indicating where a record associated with the file system element is located within a collection of records, wherein the record includes metadata related to stored data to be used to restore the file system element; and using the determined offset to retrieve the record from the collection of records" as recited in independent claims 1, 20 and 21 was reduced to

actual practice in the source code files associated with identifier "LGTpa45351" on or before October 29, 2002.

4. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.


Ajay Pratap Singh Kushwah


Date

Venkatesha Murthy

Date

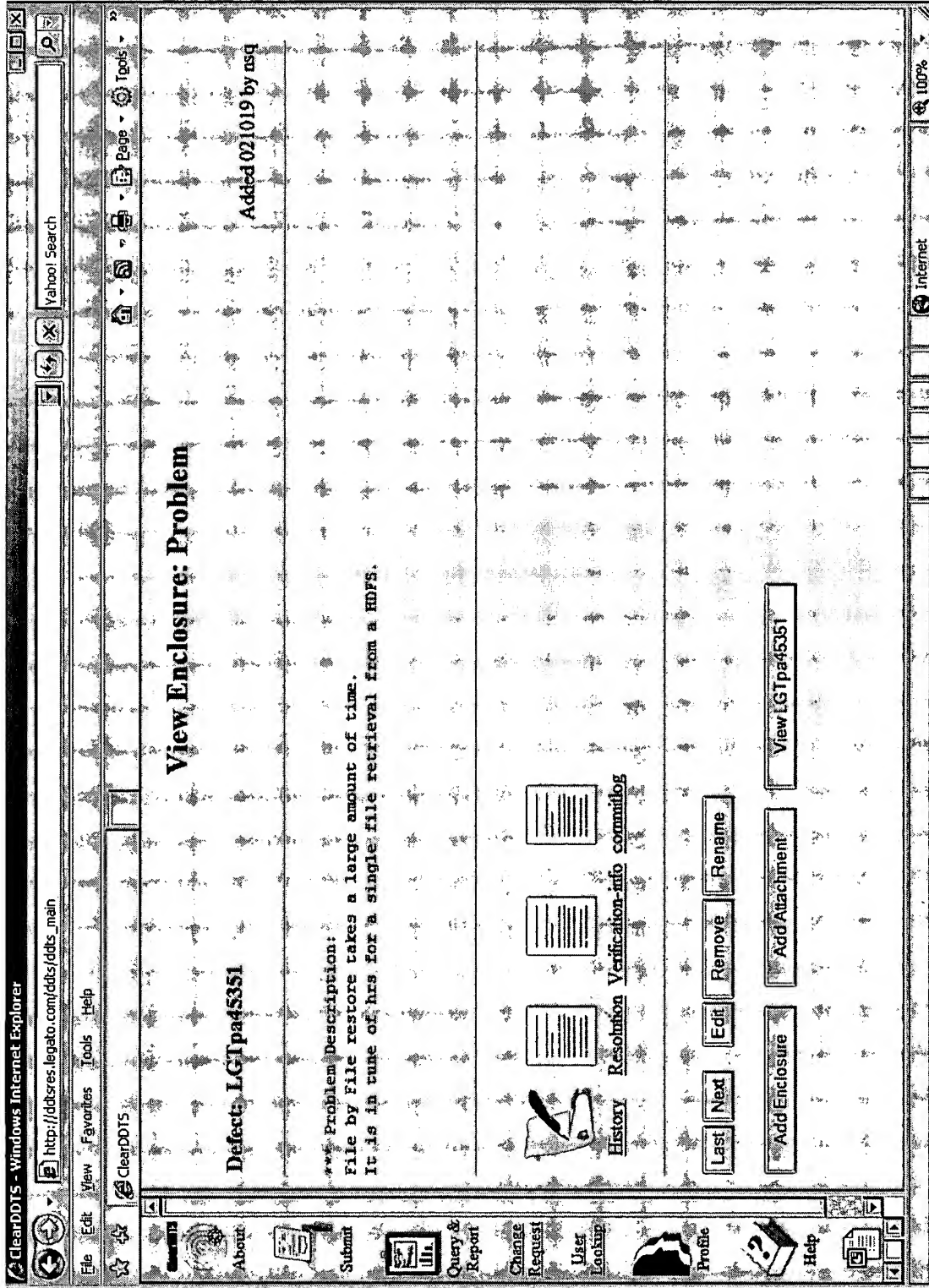


Exhibit A

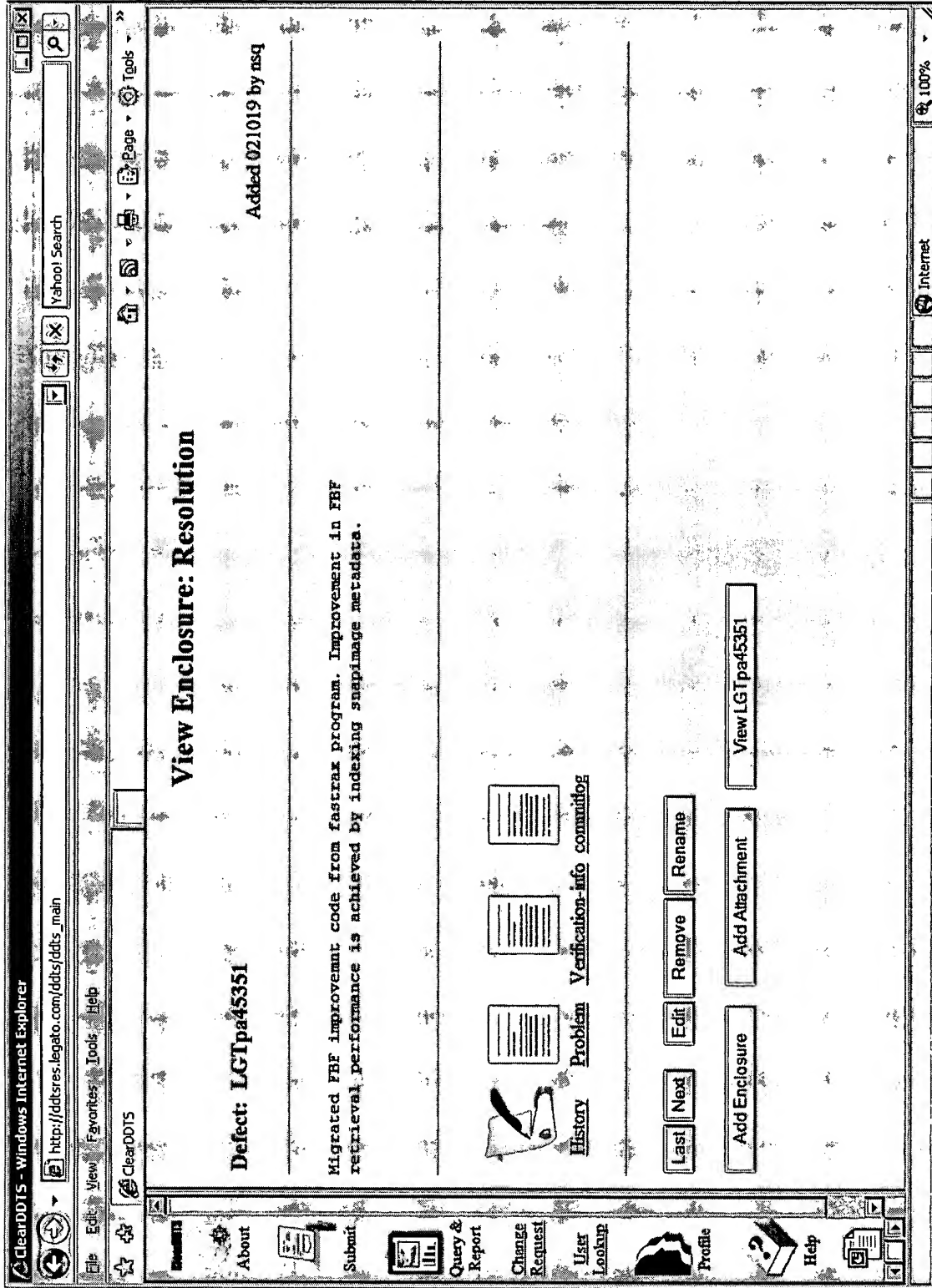


Exhibit B

Exhibit D - Differences between versions 1.5.16.2 and 1.5.16.3 of BigCacheInterfaces.cpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\bigimpl>x:\
cvs\cvs.exe diff -r 1.5.16.2 -r 1.5.16.3 BigCacheInterfaces.cpp
Index: BigCacheInterfaces.cpp
```

```
=====
===
```

```
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/bigimpl
/BigCacheInterfaces.cpp,v
retrieving revision 1.5.16.2
retrieving revision 1.5.16.3
diff -r1.5.16.2 -r1.5.16.3
2c2
< static char rcsid[] = "@(#) $Id: BigCacheInterfaces.cpp,v
1.5.16.2 2001/02/09 09:22:05 nsq Exp $";
---
> static char rcsid[] = "@(#) $Id: BigCacheInterfaces.cpp,v
1.5.16.3 2002/10/19 22:13:12 nsq Exp $";
8a9,11
> // Revision 1.5.16.3 2002/10/19 22:13:12 nsq
> // LGTpa45351: generates a index file for snapimage metadata
> //
132a136,170
>     if (type == INODE_INDEX_CACHE_FILE) {
>         if ((file = OpenLogFile("mdcache",
"celestra.inode_index", 7)) == NULL) {
>             Error(I18N(50, "Failed to open indoe index cache
file."));
>             return (-1);
>         }
>         DebugPD(ASCII("Inode Index Data file =%s"),
LogFileName);
>         inodeindexCacheFilename = strdup(LogFileName);
>         if ((cp = strrchr(LogFileName, '/')) == NULL) {
>             Error(I18N(48, "Invalid Cache file name: %s"),
LogFileName);
>             return (-1);
>         }
>         if ((ret = mgrPtr->addEnv("INODE_INDEX_CACHEID", cp +
1)) != IGSError_NONE) {
>             Error(I18N(49, "Failed to addEnv for %s"), cp + 1);
>             return (-1);
>         }
>         return (fileno(file));
```

```

>     }
>     if (type == MD_INODE_CACHE_FILE) {
>         if ((file = OpenLogFile("mdcache", "celestra.mdinodes",
7)) == NULL) {
>             Error(I18N(50, "Failed to open mdinodes cache
file."));
>             return (-1);
>         }
>         DebugPD(ASCII("MdInode Data file =%s"), LogFileName);
>         mdinodeCacheFilename = strdup(LogFileName);
>         if ((cp = strrchr(LogFileName, '/')) == NULL) {
>             Error(I18N(48, "Invalid Cache file name: %s"),
LogFileName);
>             return (-1);
>         }
>         if ((ret = mgrPtr->addEnv("MD_INODE_CACHEID", cp + 1))
!= IGSError_NONE) {
>             Error(I18N(49, "Failed to addEnv for %s"), cp + 1);
>             return (-1);
>         }
>         return (fileno(file));
>     }
>

```

Exhibit E – Differences between versions 1.16.6.15 and 1.16.6.16 of CelestraBigImpl.cpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\bigimpl>x:\
cvs\cvs.exe diff -r 1.16.6.15 -r 1.16.6.16 CelestraBigImpl.cpp
Index: CelestraBigImpl.cpp
```

```
=====
===
```

```
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/bigimpl
/CelestraBigImpl.cpp,v
retrieving revision 1.16.6.15
retrieving revision 1.16.6.16
diff -r1.16.6.15 -r1.16.6.16
2c2
< static char rcsid[] = "@(#) $Id: CelestraBigImpl.cpp,v
1.16.6.15 2001/10/18 12:15:12 harish Exp $";
---
> static char rcsid[] = "@(#) $Id: CelestraBigImpl.cpp,v
1.16.6.16 2002/10/19 22:13:12 nsq Exp $";
9a10,12
> // Revision 1.16.6.16 2002/10/19 22:13:12 nsq
> // LGTpa45351: generates a index file for snapimage metadata
> //
450a454,457
> //Introduced to improve performance of Metadata, Mapdata
writing to Tape
>
> #define BUFSIZEx200 200 * BUFSIZE
>
453a461
> bool generateMetaData = TRUE;
470a479,480
>     mdInodeSize = 0;
>     mdInodeIndexSize = 0;
519c529,532
<
---
>     if (inodeIndexTable != NULL) {
>         free(inodeIndexTable);
>         inodeIndexTable = NULL;
>     }
564a578,582
>         if (::getenv("NO_FH_MDG")) {
>             generateMetaData = TRUE;
>         } else {
```

```

>                                     generateMetaData = FALSE;
>                                     }
604a623,632
>         inodeIndexTableSize=10000;
>
>         if ((inodeIndexTable = (InodeIndexRec
*)malloc(inodeIndexTableSize*sizeof(InodeIndexRec))) == NULL)
>         {
>             retVal= new IGSError(-1, I18N(12, "malloc failed."));
>             goto done;
>         }
>         inodellimit=0;
>         inodeulimit=inodeIndexTableSize;
>         memset(inodeIndexTable, 0,
inodeIndexTableSize*sizeof(InodeIndexRec));
624a653,654
>         //char inodeIndexFilename[MAXPATHLEN];
>         // char mdDirName[MAXPATHLEN];
634c664
<         FILE *handle;
---
>         //FILE *handle;
646a677
>         inodeIndexFd = -1;
887,894c918,920
<         sprintf(mdInodeFilename, "%s/mdcache/mdinodes.%d-%d",
AppHome, pid, callCount);
<
< #ifdef DM_WINDOWS_NT
<         if ((handle = fopen(mdInodeFilename, "w+b")) == NULL) {
<             DebugPI(ASCII("Cannot open metadata dir buffer file %s:
%s \n"),
<                     mdInodeFilename,
<                     ErrorMsg(errno));
<             retVal = new IGSError(-1, I18N(59, "Cannot open metadata
Inode buffer file: %s"), ErrorMsg(errno));
---
>         if ((mdInodeFd = mcf_open(MD_INODE_CACHE_FILE)) == -1) {
>             retVal = new IGSError(-1,
>                                     I18N(18, "Failed to open Inode cache
file"));
896,897d921
<         } else {
<             mdInodeFd = fileno(handle);
899d922
< #else
901,904c924,930

```

```

<      if ((mdInodeFd = open(mdInodeFilename, O_CREAT | O_RDWR |
O_TRUNC, 0600)) < 0) {
<          DebugPI(ASCII("Cannot open metadata dir buffer file %s:
%s \n"),
<                  mdInodeFilename,
<                  ErrorMsg(errno));
---
>      // write the metadata header
>      if ((retVal = WriteMetaDataHdr(mdInodeFd, fsName)) !=
IGSError_NONE)
>          goto done;
>      else
>          DebugPD(ASCII("Metadata header written"));
>
>      if ((inodeIndexFd = mcf_open(INODE_INDEX_CACHE_FILE)) == -
1) {
906c932
<                  I18N(20, "Cannot open metadata
Inode buffer file: %s"), ErrorMsg(errno));
---
>                  I18N(18, "Failed to open Inode Index cache
file"));
910d935
< #endif
912,914c937
<      if (!DebugPD(ASCII("keeping mdInodeFilename = %s"),
mdInodeFilename) &&
<          (unlink(mdInodeFilename) < 0))
<          DebugDS(ASCII("Cannot unlink file: %s (%s)"),
mdInodeFilename, ErrorMsg(errno));
---
>
986c1009,1030
<      DebugPD(ASCII("Metadata trailer written"));
---
>      else
>          DebugPD(ASCII("File MD Inode Metadata trailer
written"));
>
>          DebugPD(ASCII("inodeIndexFd = %d\n"), inodeIndexFd);
>          if (write(inodeIndexFd,
inodeIndexTable, inodeIndexTableSize*sizeof(InodeIndexRec)) < 0 )
{
>              retVal = new IGSError(-1,
>                  I18N(42, "Cannot write inodeindextable to
InodeIndexFile : %s"), ErrorMsg(errno));
>              DebugPI(ASCII("%s"), retVal->getMessage());

```

```

>         DebugPD(ASCII("Write Error inodeIndexFd file"));
>         goto done;
>     }
>     // write the Dir metadata trailer
>     if ((retVal = WriteMetaDataTrailer(mdDirFd)) !=
IGSError_NONE)
>         goto done;
>     else
>         DebugPD(ASCII("Dir Metadata trailer written"));
>
>
>     if ((retVal = WriteMetaDataTrailer(inodeIndexFd)) !=
IGSError_NONE)
>         goto done;
>     else
>         DebugPD(ASCII("INODE INDEX Metadata trailer
written"));
1159a1204,1205
>     if (inodeIndexFd >= 0)
>         mcf_destroy(inodeIndexFd);
1163a1210,1211
>     if (inodeIndexFd >= 0)
>         mcf_destroy(inodeIndexFd);
1257a1306,1309
>     if (inodeIndexTable != NULL) { //PURIFY reported leak
>         free(inodeIndexTable);
>         inodeIndexTable = NULL;
>     }
1694c1746
<     char *buffer = (char *) Malloc(BUFSIZE);
---
>     char *buffer = (char *) Malloc(BUFSIZE*200);
1696a1749
>     char strToAdd[20];
1702c1755
<
---
> Log(I18N(-1,"Writing Metadata to Tape - %x"),retVal);
1722a1776,1781
>     if (lseek(inodeIndexFd, 0, SEEK_SET) != 0) {
>         retVal = new IGSError(-1,
>             I18N(25, "Error seeking to
begining of inode data: %s"), ErrorMessage(errno));
>         DebugPI(ASCII("%s"), retVal->getMessage());
>         goto done;
>     }
1728c1787

```

```

<          readSz = mcf_read(mdDirFd, buffer, BUFSIZE);
---
>          readSz = mcf_read(mdDirFd, buffer, BUFSIZEx200);
1737c1796
<          } else if (readSz < BUFSIZE) {
---
>          } else if (readSz < BUFSIZEx200) {
1742c1801
<          retVal = imageFmtServices->writeFileData(buffer,
BUFSIZE);
---
>          retVal = imageFmtServices->writeFileData(buffer,
BUFSIZEx200);
1751c1810,1815
<          while (readSz == BUFSIZE);
---
>          while (readSz == BUFSIZEx200);
>          mdDirSize = mdSize;
>          sprintf(strToAdd, "%lld", mdDirSize);
>          if ((retVal = mgrPtr->addEnv("DIR_METADATA_SIZE",
strToAdd)) != IGSError_NONE)
>              goto done;
>          DebugPD(ASCII("Dir METADATA SIZE = %lld\n"), mdDirSize);
1764c1828
<          readSz = read(mdInodeFd, buffer + tempSz, BUFSIZE -
tempSz);
---
>          readSz = read(mdInodeFd, buffer + tempSz,
BUFSIZEx200 - tempSz);
1773c1837
<          } else if (readSz < BUFSIZE - tempSz) {
---
>          } else if (readSz < BUFSIZEx200 - tempSz) {
1776c1840,1841
<          memset((buffer + readSz + tempSz), 0, BUFSIZE -
(readSz + tempSz));
---
>          // memset((buffer + readSz + tempSz), 0, BUFSIZE
- (readSz + tempSz));
>          mdSize += readSz;
1779c1844
<          if ((retVal = imageFmtServices-
>writeFileData(buffer, BUFSIZE)) != IGSError_NONE)
---
>          if ((retVal = imageFmtServices-
>writeFileData(buffer, BUFSIZEx200)) != IGSError_NONE)
1782c1847

```



```

<          if (mcf_write(mdDirFd, buffer + tempSz, BUFSIZE -
tempSz) < 0) {
---
>          /*      if (mcf_write(mdDirFd, buffer + tempSz, BUFSIZE -
tempSz) < 0) {
1791a1857,1858
> */
>
1798c1865
<          while (readSz == BUFSIZE);
---
>          while (readSz == BUFSIZEEx200);
1799a1867,1913
>          mdInodeSize = mdSize - mdDirSize;
>          sprintf(strToAdd, "%lld", mdInodeSize);
>          if ((retVal = mgrPtr->addEnv("INODE_METADATA_SIZE",
strToAdd)) != IGSError_NONE)
>              goto done;
>          DebugPD(ASCII("Inode METADATA SIZE = %lld\n"),
mdInodeSize);
>          tempSz = readSz+tempSz;
>          // write inode index to tape and append it
>          // over the inode metadata buffer file
>          do {
>              // write the inode index metadata to the tape
>              // tempSz will only be usefule in the forst read
>              // it is added to check the case when the previous
md read
>              // is a partial buffer.
>
>              readSz = read(inodeIndexFd, buffer + tempSz,
BUFSIZEEx200 - tempSz);
>              if (readSz < 0) {
>                  retVal = new IGSError(-1,
>                                      I18N(27, "Metadata inode
buffer read failed: %s"), ErrorMessage(errno));
>                  DebugPI(ASCII("%s"), retVal->getMessage());
>                  goto done;
>              }
>              if (readSz == 0) {
>                  DebugPI(ASCII("AppendMetadataToBackupImage:
Metadata inode reads done"));
>              } else if (readSz < BUFSIZEEx200 - tempSz) {
>                  DebugPI(ASCII("AppendMetadataToBackupImage:
Incomplete Read from metadata inode buffer"));
>                  // zero out rest of the buffer

```

```

>         memset((buffer + readSz + tempSz), 0,
BUFSIZEEx200 - (readSz + tempSz));
>     }
>     DebugPI(ASCII("Writing metadata %x size "), *(long
*) buffer);
>     if ((retVal = imageFmtServices-
>writeFileData(buffer, BUFSIZEEx200)) != IGSError_NONE)
>         goto done;
>
>
>         mdSize += readSz;
>         if (tempSz) {
>             readSz += tempSz;
>             tempSz = 0;
>         }
>     }
>     while (readSz == BUFSIZEEx200);
>
>     mdInodeIndexSize = mdSize - mdDirSize - mdInodeSize;
>     sprintf(strToAdd, "%lld", mdInodeIndexSize);
>     if ((retVal = mgrPtr-
>addEnv("INODEINDEX_METADATA_SIZE", strToAdd)) != IGSError_NONE)
>         goto done;
>     DebugPD(ASCII("Inode Index METADATA SIZE = %lld\n"),
mdInodeIndexSize);
>
1805a1920
>     Log(I18N(-1, "Copied Metadata to Tape - %x"), retVal);
1834c1949
<     buffer = (char *) Malloc(BUFSIZE);
---
>     buffer = (char *) Malloc(BUFSIZEEx200);
1840c1955,1956
<     if (mapCacheFileIncomplete == false) {
---
>     Log(I18N(-1, "Writing Mapdata to Tape - %x"), retVal);
>     if (mapCacheFileIncomplete == false) {
1849c1965
<         readSz = mcf_read(mapFd, buffer, BUFSIZE);
---
>         readSz = mcf_read(mapFd, buffer, BUFSIZEEx200);
1857,1858c1973,1974
<         if (readSz < BUFSIZE) {
<             memset((buffer + readSz), 0, BUFSIZE - readSz);
---
>         if (readSz < BUFSIZEEx200) {

```

```

>          memset((buffer + readSz), 0, BUFSIZEx200 -
readSz);
1869c1985
<          retVal = imageFmtServices->writeFileData(buffer,
BUFSIZE);
---
>          retVal = imageFmtServices->writeFileData(buffer,
BUFSIZEx200);
1880c1996
<          while (readSz == BUFSIZE);
---
>          while (readSz == BUFSIZEx200);
1888a2005
>          Log(I18N(-1,"Copied Mapdata to Tape -
%x"),retVal);
2114c2231
< CelestraBigImpl::WriteMetaDataHdr(int mdDirFd, char
*srcDevice)
---
> CelestraBigImpl::WriteMetaDataHdr(int mdFd, char *srcDevice)
2128c2245,2250
<      sidfField.setFidNumber(METADATA_HEADER);
---
>      if (mdFd == mdDirFd )
>          sidfField.setFidNumber(DIR_METADATA_HEADER);
>      else if (mdFd == mdInodeFd )
>          sidfField.setFidNumber(MDINODE_METADATA_HEADER);
>      else if (mdFd == inodeIndexFd )
>          sidfField.setFidNumber(INODE_INDEX_METADATA_HEADER);
2165c2287
<      if (mcf_write(mdDirFd, buff, hdrSize) < 0) {
---
>      if (mcf_write(mdFd, buff, hdrSize) < 0) {
2439c2561,2566
<      sidfField.setFidNumber(METADATA_TRAILER);
---
>      if (inoFd == mdDirFd )
>          sidfField.setFidNumber(DIR_METADATA_TRAILER);
>      else if (inoFd == mdInodeFd )
>          sidfField.setFidNumber(MDINODE_METADATA_TRAILER);
>      else if (inoFd == inodeIndexFd )
>          sidfField.setFidNumber(INODE_INDEX_METADATA_TRAILER);
2524a2652,2655
>      if ((retVal =
updateInodeIndexTable(inodeInfo.inoNum, mdDirFd)) !=
IGSError_NONE) {

```

```

>         DebugPD(ASCII("SendAttrStream: Error in Updating
InodeIndexTable for Dir Inode"));
>         goto done;
>     }
2556a2688,2691
>     if ((retVal =
updateInodeIndexTable(inodeInfo.inoNum, mdInodeFd)) !=
IGSError_NONE) {
>         DebugPD(ASCII("SendAttrStream: Error in Updating
InodeIndexTable for File Inode"));
>         goto done;
>     }
2789a2925,2956
>
> IGSError *CelestraBigImpl::updateInodeIndexTable(IGSino_t
inoNum, int destFd)
> {
>     struct stat stat_buf;
>     __int64 offsetInFile=0;
>     IGSError *retVal = IGSError_NONE;
>
>     fstat(destFd, &stat_buf);
>     offsetInFile = stat_buf.st_size;
>     //DebugPD(ASCII("Node Real Offset number offset=%d\n"),
offsetInFile);
>     while (inoNum >= inodeulimit )
>     {
>         if (write(inodeIndexFd,
inodeIndexTable,inodeIndexTableSize*sizeof(InodeIndexRec)) < 0)
>         {
>             retVal = new IGSError(-1,
>                                     I18N(42, "Cannot write
inodeindextable to InodeIndexFile : %s"), ErrorMessage(errno));
>             DebugPI(ASCII("%s"), retVal->getMessage());
>         }
>     }
>
>     memset(inodeIndexTable,0,inodeIndexTableSize*sizeof(InodeIndexRe
c));
>     inodellimit = inodeulimit;
>     inodeulimit += inodeIndexTableSize;
> }
> if (destFd == mdDirFd) {
>     inodeIndexTable[inoNum -
inodellimit].typeAndOffset=offsetInFile | DIR_BIT_MASK;
> } else {
>     inodeIndexTable[inoNum -
inodellimit].typeAndOffset=offsetInFile ;

```

```
>     }
>     //DebugPD(ASCII("before bit shifts ino=%d,offset=%d\n"),
inoNum, inodeIndexTable[inoNum - inodellimit].typeAndOffset);
>     //offsetInFile = offsetInFile & 0x7fffffff ;
>     //DebugPD(ASCII("after bit shifts ino=%d,offset=%d\n"),
inoNum, offsetInFile);
>     return (retVal);
> }
>
```

Exhibit F – Differences between versions 1.5.16.3 and 1.5.16.4 of CelestraBigImpl.hpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\bigimpl>x:\
cvs\cvs.exe diff -r 1.5.16.3 -r 1.5.16.4 CelestraBigImpl.hpp
Index: CelestraBigImpl.hpp
```

```
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/bigimpl
/CelestraBigImpl.hpp,v
retrieving revision 1.5.16.3
retrieving revision 1.5.16.4
diff -r1.5.16.3 -r1.5.16.4
5c5
< // $Id: CelestraBigImpl.hpp,v 1.5.16.3 2001/05/01 09:51:39
harish Exp $
---
> // $Id: CelestraBigImpl.hpp,v 1.5.16.4 2002/10/20 09:53:33 nsq
Exp $
55a56,61
> #ifndef INODE_INDEX_CACHE_FILE
> #define INODE_INDEX_CACHE_FILE      (103)
> #endif
> #ifndef MD_INODE_CACHE_FILE
> #define MD_INODE_CACHE_FILE        (104)
> #endif
97a104
>     IGSError *updateInodeIndexTable(ino_t inoNumber, int
destFd);
104a112
>     IGSError *updateInodeIndexTable(IGSino_t inoNumber, int
destFd);
141a150,152
>     long long mdDirSize;           /* size of Dir metadata */
>     long long mdInodeSize;         /* size of File metadata */
>     long long mdInodeIndexSize;    /* size of Inode Index
metadata */
150a162,164
>     quad mdDirSize;               /* size of Dir metadata */
>     quad mdInodeSize;             /* size of File metadata */
>     quad mdInodeIndexSize;        /* size of Inode Index
metadata */
156a171
>     int inodeIndexFd;              /* inode index file fd */
159a175,176
```

```

>     char *inodeindexCacheFilename;
>     char *mdinodeCacheFilename;
185a203,212
>     InodeIndexRec *inodeIndexTable;
> #ifndef DM_WINDOWS_NT
>     long long inodeIndexTableSize;
> #else
>     quad inodeIndexTableSize;
> #endif
>     //int newRestoreDesign;
>     int inodellimit;
>     int inodeulimit;
>
194a222,224
>     IGSError *AppendDirMetadataToBackupImage(void);
>     IGSError *AppendMdInodeMetadataToBackupImage(void);
>     IGSError *AppendInodeIndexMetadataToBackupImage(void);

```

Exhibit G – Differences between versions 1.1.2.1 and 1.1.2.2 of MDImage.cpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.1 -r 1.1.2.2 MDImage.cpp
Index: MDImage.cpp
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/MDImage.cpp,v
retrieving revision 1.1.2.1
retrieving revision 1.1.2.2
diff -r1.1.2.1 -r1.1.2.2
2c2
< #ident "$Id: MDImage.cpp,v 1.1.2.1 2001/02/10 09:41:25 nsq Exp
$ Copyright (c) 2001, Legato Systems, Inc."
---
> #ident "$Id: MDImage.cpp,v 1.1.2.2 2002/10/19 22:26:54 nsq Exp
$ Copyright (c) 2002, Legato Systems, Inc."
6c6
< * Copyright (c) 2001, Legato Systems, Inc.
---
> * Copyright (c) 2002, Legato Systems, Inc.
10,12d9
< #if !defined(lint) && !defined(SABER)
< static char rcsid[] = "@(#) $Id: MDImage.cpp,v 1.1.2.1
2001/02/10 09:41:25 nsq Exp $";
< #endif
15d11
< * Copyright (c) 2001, Legato Systems Incorporated.
17a14,16
> * Revision 1.1.2.2 2002/10/19 22:26:54 nsq
> * LGTpa45351: added code to use indexing of Metadata for FBF
retrieval
> *
86c85,87
< // #include "sysfiles.h"
---
> #include <stdlib.h>
> #include <stdio.h>
> #include "sysfiles.hpp"
240a242,342
>
> /* Added for new restore design */
> NewMDImage::NewMDImage(int mdImageSize, int argImageHandle)
```



```

> {
>     imageHandle = argImageHandle;
>     imageSize = mdImageSize;
>     DebugPD(ASCII("imageHandle = %d, mdSize =%d \n"),
imageHandle, imageSize);
>     lseek(imageHandle, 0, SEEK_SET);
> }
>
> /*
>  * read
>  -----
>  -----
>  * Function      :
>  *
>  * Procedure     :
>  * Inputs        :
>  * Params:
>  * dataBuffer - data buffer
>  * numBytes    - number of bytes to be read.
>  *
>  * Outputs       : bytes Read. -1 on error.
>  * Messages      :
>  * Side Effects:
>  * Bugs :
>  * History       :
>  -----
>  -----
>  */
> ssize_t
> NewMDImage::read(char *dataBuffer, ssize_t numBytes)
> {
>     if (::read(imageHandle, dataBuffer, numBytes) < numBytes )
>     {
>         Error(I18N(-1, "Error reading Metadata File"));
>         return (-1);
>     }
>     return numBytes;
> }
> /*
>  * write
>  -----
>  -----
>  * Function      :
>  *
>  * Procedure     :
>  * Inputs        :
>  * Params:

```

```

> * dataBuffer - data buffer
> * numBytes   - number of bytes to be read.
> *
> * Outputs      : bytes Written. -1 on error.
> * Messages     :
> * Side Effects:
> * Bugs :
> * History      :
> -----
-----
> */
>
> ssize_t
> NewMDImage::write(char *dataBuffer, ssize_t numBytes)
> {
>     // later
> #ifdef DM_WINDOWS_NT
>     return (ssize_t) 1;
> #endif
> }
>
> off_t
> NewMDImage::seek(off_t offset, int whence)
> {
>     DebugPD(ASCII("Imagehandle = %d, offset=%d\n"),
imageHandle, offset);
>     return lseek(imageHandle, offset, whence);
> }
>
> int
> NewMDImage::copyMetadataToDisk(char *mdFileName)
> {
>     char *mdBuffer;
>     int dirMetadataFd = -1;
>
>     DebugPD("Metadata file name on disk is %s", mdFileName);
>     imageHandle = open(mdFileName, O_WRONLY | O_TRUNC |
O_CREAT | O_BINARY, 0666);
>     if ((mdBuffer = (char *) malloc(imageSize)) == NULL) {
>         setError(new IGSError(-1, I18N(12, "malloc failed.")));
>         return (-1);
>     }
>     if (getFileData(mdBuffer, imageSize) < imageSize) {
>         setError(new IGSError(-1,
I18N(2, "Error in reading metadata from
tape:%s"), ErrorMessage(errno)));
>         return (-1);

```

```

>     }
>     DebugPD("Copying Metadata from tape to disk FD %d =%s",
mdSaveFd);
>     if (::write(imageHandle, mdBuffer, imageSize) < 0) {
>         setError(new IGSError(-1,
>             I18N(3, "Could not copy Metadata from tape to
disk"), ErrorMessage(errno)));
>         free(mdBuffer);
>         return (-1);
>     }
>     DebugPD(ASCII("VVM TMP: Freeing mdBuffer\n"));
>     free(mdBuffer);
>     return (1);
> }
>

```

Exhibit H – Differences between versions 1.1.2.1 and 1.1.2.2 of MDImage.hpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.1 -r 1.1.2.2 MDImage.hpp
Index: MDImage.hpp
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/MDImage.hpp,v
retrieving revision 1.1.2.1
retrieving revision 1.1.2.2
diff -r1.1.2.1 -r1.1.2.2
1c1
< /* $Id: MDImage.hpp,v 1.1.2.1 2001/02/10 09:41:25 nsq Exp $
Copyright (c) 2001, Legato Systems, Inc. */
---
> /* $Id: MDImage.hpp,v 1.1.2.2 2002/10/19 22:26:54 nsq Exp $
Copyright (c) 2002, Legato Systems, Inc. */
4c4
< * Copyright (c) 2001, Legato Systems, Inc.
---
> * Copyright (c) 2002, Legato Systems, Inc.
13a14,16
> * Revision 1.1.2.2 2002/10/19 22:26:54 nsq
> * LGTpa45351: added code to use indexing of Metadata for FBF
retrieval
> *
122a126,139
> class NewMDImage:public File
> {
>     int imageHandle;
>     int imageSize;
>
>     public:
>     NewMDImage(int imageSize, int argImageHandle = -1);
>     ~NewMDImage(void) {
>     } ssize_t write(char *buffer, ssize_t numBytes);
>     ssize_t read(char *buffer, ssize_t numBytes);
>     off_t seek(off_t offset, int whence);
>     int copyMetadataToDisk(char *mdFileName);
> };
>
```

Exhibit I – Differences between versions 1.1.2.1 and 1.1.2.2 of getnext.hpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.1 -r 1.1.2.2  getnext.hpp
Index: getnext.hpp
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/getnext.hpp,v
retrieving revision 1.1.2.1
retrieving revision 1.1.2.2
diff -r1.1.2.1 -r1.1.2.2
1c1
< /* $Id: getnext.hpp,v 1.1.2.1 2001/02/10 09:41:25 nsq Exp $
Copyright (c) 2001, Legato Systems, Inc. */
---
> /* $Id: getnext.hpp,v 1.1.2.2 2002/10/19 22:33:05 nsq Exp $
Copyright (c) 2002, Legato Systems, Inc. */
23c23
< extern MDImage* mdImage;
---
> extern File* mdImage;
```

Exhibit J – Differences between versions 1.1.2.14 and 1.1.2.15 of rip.cpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.14 -r 1.1.2.15  rip.cpp
Index: rip.cpp
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/rip.cpp,v
retrieving revision 1.1.2.14
retrieving revision 1.1.2.15
diff -r1.1.2.14 -r1.1.2.15
2c2
< #ident "$Id: rip.cpp,v 1.1.2.14 2001/09/28 07:00:11 yogita Exp
$ Copyright (c) 2001, Legato Systems, Inc."
---
> #ident "$Id: rip.cpp,v 1.1.2.15 2002/10/19 22:29:02 nsq Exp $
Copyright (c) 2002, Legato Systems, Inc."
11c11
< static char rcsid[] = "@(#) $Id: rip.cpp,v 1.1.2.14 2001/09/28
07:00:11 yogita Exp $";
---
> static char rcsid[] = "@(#) $Id: rip.cpp,v 1.1.2.15 2002/10/19
22:29:02 nsq Exp $";
17a18,20
> * Revision 1.1.2.15 2002/10/19 22:29:02 nsq
> * LGTPa45351: added code to use indexing of metadata for FBF
retrieval
> *
428a432
> #include "rtrvtree.h"
479a484,486
> int newRestoreDesign =1;
> struct InodeIndexRec *inodeIndexTable;
> int inodeIndexFile = -1;
484c491,492
< MDImage *mdImage;
---
> File *mdImage;
> File *fileMdImage;
510c518
< long dataSize = MD_TAPEBUFSIZE * 200;
---
> long dataSize = MD_TAPEBUFSIZE * 200, perftime;
```

```

555c563
<
---
> Log(I18N(3,"Restoring DDIMAGE"));
662c670
<
---
> perftime = time (0);
718a727,732
>             if ( (time(0) - perftime) > 600 ) {
>                 perftime = time (0);
>                 Log(I18N(3,"Restored = %ld GB" ),(int)
(((__int64)blk * 512)/ ((__int64)1024*1024*1024)) );
>
>             }
>
783,787c797
< #ifndef CRM_ENABLED
< StartRetrieval(int imageFormat, CelestraCount_t copySize,
CelestraCount_t bcLimitCount)
< #else
< StartRetrieval(int imageFormat, CrmFsys * rcf,
CelestraCount_t copySize, CelestraCount_t bcLimitCount)
< #endif
---
> StartRetrieval(int imageFormat, CelestraCount_t copySize,
CelestraCount_t bcLimitCount)
789c799
<     int mdFile = -1, mapFile = -1;
---
>     int mdFile = -1, tmpmdFile = -1, mapFile = -1,
MdInodeFile = -1, tmpFile = -1;
790a801
>     char tmpmdfname[DM_MAXPATHLEN];
791a803,806
>     char *InodeIndexId = NULL;
>     char *mdInodeId = NULL;
>     char inodeIndexfname[DM_MAXPATHLEN];
>     char mdInodeFileName[DM_MAXPATHLEN];
816c831
<
---
>     //Sleep(2 *60 * 1000);
838,842d852
< #ifdef CRM_ENABLED
<     if (rcf == NULL) {

```

```

<          DebugPD(I18N(20, "CrmFsys object is not
presented."));
<      }
< #endif
853,862c863,866
<      mgr_getEnv("MAP_CRMID", &mapId);
<      mgr_getEnv("MD_CRMID", &mdId);
<      if (mapId == NULL || mdId == NULL) {
<          DebugDI("Failed to get env  for MD/MAP");
<          if (mapId != NULL)
<              Free(mapId);
<          if (mdId != NULL)
<              Free(mdId);
<          mgr_getEnv("MD_CACHEID", &mdId);
<          mgr_getEnv("MAP_CACHEID", &mapId);
---
>      mgr_getEnv("MD_CACHEID", &mdId);
>      mgr_getEnv("MAP_CACHEID", &mapId);
>
>      if (newRestoreDesign == 0) {
870c874
<          if ((mdFile = open(mdfname, O_RDONLY |
O_BINARY)) >= 0 &&
---
>          if ((mdFile = open(mdfname, O_RDONLY |
O_BINARY)) >= 0 &&
878,906d881
<      } else {
< #ifdef CRM_ENABLED
<      if (rcf == NULL) {
< #endif
<          DebugDI("CrmFsys is not available, trying to get
MDCACHE ID...");
<          if (mapId != NULL)
<              Free(mapId);
<          if (mdId != NULL)
<              Free(mdId);
<
<          mgr_getEnv("MD_CACHEID", &mdId);
<          mgr_getEnv("MAP_CACHEID", &mapId);
<
<          if (mdId == NULL || mapId == NULL) {
<              mdFile = mapFile = -1;
<          } else {
<              sprintf(mdfname, "%s/mdcache/%s", AppHome,
mdId);

```



```

<          sprintf(mapfname, "%s/mdcache/%s", AppHome,
mapId);
<          DebugUI("md cache filename =%s", mdfname);
<          DebugUI("map cache filename =%s", mapfname);
<          if ((mdFile = open(mdfname, O_RDONLY |
O_BINARY)) >= 0 &&
<              (mapFile = open(mapfname, O_RDONLY |
O_BINARY)) < 0) {
<              close(mdFile);
<              mdFile = -1;
<          } else {
<              useMdID = DM_TRUE;
<          }
<      }
< #ifdef CRM_ENABLED
908,920c883,915
<          mapIdN = atol(mapId);
<          mdIdN = atol(mdId);
<
<          if ((mdFile = rcf->crmf_open(mdIdN, CRMF_READ)) >= 0
&&
<              (mapFile = rcf->crmf_open(mapIdN, CRMF_READ)) <
0) {
<              rcf->crmf_close(mdFile);          /* either open
both or neither */
<              mdFile = -1;
<          } else {
<              useCrmID = DM_TRUE;
<          }
<      }
< #endif
<      }
---
>          mgr_getEnv("INODE_INDEX_CACHEID",
&InodeIndexId);
>          mgr_getEnv("MD_INODE_CACHEID", &mdInodeId);
>          if ((mdId == NULL) || (mapId == NULL) ||
(InodeIndexId == NULL) || (mdInodeId == NULL)) {
>              mdFile = -1;
>          } else {
>              dm_bool metDataFilesOnDisk=DM_FALSE;
>              sprintf(mdfname, "%s/mdcache/%s",
AppHome, mdId);
>              sprintf(tmpmdfname, "%s/mdcache/tmp_%s",
AppHome, mdId);
>              sprintf(inodeIndexfname,
"%s/mdcache/%s", AppHome, InodeIndexId);

```

```

>             sprintf(mdInodeFileName,
"%s/mdcache/%s", AppHome, mdInodeId);
>             sprintf(mapfname, "%s/mdcache/%s",
AppHome, mapId);
>             DebugUI("md cache filename =%s",
mdfname);
>             DebugUI("file metadata cache filename
=%s", mdInodeFileName);
>             DebugUI("Inode Index cache filename
=%s", inodeIndexfname);
>             DebugUI("map cache filename =%s",
mapfname);
>             if ( ((mdFile = open(mdfname,
O_RDONLY | O_BINARY)) >= 0) &&
>                 ((inodeIndexFile =
open(inodeIndexfname, O_RDONLY | O_BINARY)) >= 0) &&
>                 ((mapFile =
open(mapfname, O_RDONLY | O_BINARY)) >= 0) &&
>                 ((MdInodeFile =
open(mdInodeFileName, O_RDONLY | O_BINARY)) >= 0)) {
>                 metaDataFilesOnDisk = DM_TRUE;
>                 DebugPD(ASCII("All metadata
files are on disk\n"));
>                 useMdID = DM_TRUE;
>             } else {
>                 /*
>                 * for now taking the approach
that metadata files are a must on the disk
>                 */
>                 Error(I18N(-1, "Could not open
metadata files.));
>                 retVal = DM_ERROR;
>                 goto done;
>             }
>
>             } /* mdFile != -1 */
>         } /* new restore design */
941,945d935
< #ifdef CRM_ENABLED
<     IDType mdn;
<     char *mdStr;
<     if (rcf == NULL) {
< #endif
958,960c948,950
<             Error(I18N(30, "Cannot allocate
memory.));
<             retVal = DM_ERROR;

```

```

<                                goto done;
---
>                                Error(I18N(30, "Cannot allocate
memory.));
>                                retVal = DM_ERROR;
>                                goto done;
962,965c952,955
<                                strcpy(mdId, "incr.md");
<                                sprintf(mdSaveFile, "%s/mdcache/%s", AppHome,
mdId);
<                                sprintf(mdcacheDir, "%s/mdcache", AppHome);
<                                if (LSTAT(mdcacheDir, &stBuf) == -1) {
---
>                                strcpy(mdId, "incr.md");
>                                sprintf(mdSaveFile, "%s/mdcache/%s",
AppHome, mdId);
>                                sprintf(mdcacheDir, "%s/mdcache",
AppHome);
>                                if (LSTAT(mdcacheDir, &stBuf) == -1) {
973,1002c963,980
<                                }
<                                if ((mdSaveFd = open(mdSaveFile,
O_WRONLY | O_TRUNC | O_CREAT | O_BINARY, 0666)) < 0) {
<                                Error(I18N(23, "Cannot
create cache file %s"), mdSaveFile);
<                                retVal = DM_ERROR;
<                                goto done;
<                                }
<                                strcpy(mdId, mdSaveFile);
<                                } else {
<                                if (mdFile < 0) {
<                                DebugPD(ASCII("mdId is
not NULL and mdFile is < 0"));
<                                sprintf(mdcacheDir, "%s/mdcache",
AppHome);
<                                sprintf(mdSaveFile, "%s/mdcache/%s",
AppHome, mdId);
<
<                                if (LSTAT(mdcacheDir, &stBuf) == -1) {
<                                DebugPD(ASCII("mdcache
directory doesn't exist. Creating %s\n"), mdcacheDir);
<                                Warning(I18N(29, "One of
the directories critical for Celestra execution does not exist.
Creati
ng %s\n"), mdcacheDir);
<                                if (MKDIR(mdcacheDir, 0700) ==
MKDIR_ERROR_VALUE) {

```

```

<                                     Error(I18N(23, "Cannot create
cache directory %s"), mdcacheDir);
<                                     retVal =
DM_ERROR;
<                                     goto
done;
<                                     }
<                                     }
<                                     if ((mdSaveFd = open(mdSaveFile, O_WRONLY
| O_TRUNC | O_CREAT | O_BINARY, 0666)) < 0) {
<                                     Error(I18N(23, "Cannot create cache
file %s"), mdSaveFile);
<                                     retVal = DM_ERROR;
<                                     goto done;
<                                     }
<                                     //free(mdId);
<                                     if ((mdId = (char *)
malloc(DM_MAXPATHLEN)) == NULL) {
<                                     Error(I18N(30,
"Cannot allocate memory."));
---
>                                     }
>                                     if ((mdSaveFd = open(mdSaveFile,
O_WRONLY | O_TRUNC | O_CREAT | O_BINARY, 0666)) < 0) {
>                                     Error(I18N(23, "Cannot create
cache file %s"), mdSaveFile);
>                                     retVal = DM_ERROR;
>                                     goto done;
>                                     }
>                                     strcpy(mdId, mdSaveFile);
>                                     } else {
>                                     if (mdFile < 0) {
>                                     DebugPD(ASCII("mdId is not NULL
and mdFile is < 0"));
>                                     sprintf(mdcacheDir, "%s/mdcache", AppHome);
>                                     sprintf(mdSaveFile, "%s/mdcache/%s", AppHome,
mdId);
>
>                                     if (LSTAT(mdcacheDir, &stBuf) == -1) {
>                                     DebugPD(ASCII("mdcache
directory doesn't exist. Creating %s\n"), mdcacheDir);
>                                     Warning(I18N(29, "One of the
directories critical for Celestra execution does not exist.
Creating %
s\n"), mdcacheDir);
>                                     if (MKDIR(mdcacheDir, 0700) ==
MKDIR_ERROR_VALUE) {

```

```

>                                     Error(I18N(23, "Cannot create cache
directory %s"), mdcacheDir);
1005,1006c983,988
<                                     }
<                                     strcpy(mdId, mdSaveFile);
---
>                                     }
>                                     }
>                                     if ((mdSaveFd = open(mdSaveFile, O_WRONLY |
O_TRUNC | O_CREAT | O_BINARY, 0666)) < 0) {
>                                     Error(I18N(23, "Cannot create cache file
%s"), mdSaveFile);
>                                     retVal = DM_ERROR;
>                                     goto done;
1007a990,996
>                                     //free(mdId);
>                                     if ((mdId = (char *)
malloc(DM_MAXPATHLEN)) == NULL) {
>                                     Error(I18N(30, "Cannot
allocate memory."));
>                                     retVal = DM_ERROR;
>                                     goto done;
>                                     }
>                                     strcpy(mdId, mdSaveFile);
1009,1015c998,1002
< #ifdef CRM_ENABLED
<     } else {
<         if (mdFile < 0) {
<             if ((mdSaveFd = rcf->crmf_open(-1, CRMF_RDWR,
&mdn)) < 0) {
<                 Error(I18N(22, "Could not open cache file
for writing metadata to incremental cache."));
<                 retVal = DM_ERROR;
<                 goto done;
---
>         }
>         if ((incrCacheFd = open(incrCacheFile,
>                                 O_WRONLY | O_TRUNC | O_CREAT |
O_BINARY, 0666)) < 0) {
>             Error(I18N(24, "Could not open cache
file for writing incrementals information."));
>             return (-1);
1017,1039c1004,1038
<         DebugPD(ASCII("Save md file is %s, fd is: %d"),
mdSaveFile, mdSaveFd);
<         sprintf(mdId, "%ld", mdn);
<     }

```

```

<     }
< #endif
<     if ((incrCacheFd = open(incrCacheFile,
<                             O_WRONLY | O_TRUNC | O_CREAT | O_BINARY,
0666)) < 0) {
<         Error(I18N(24, "Could not open cache file for
writing incrementals information.));
<         return (-1);
<     }
<
<     DebugPD(ASCII("incr cache file  fd: %d"), incrCacheFd);
<
<     len = strlen(mdId) + 1;
<     if (write(incrCacheFd, &len, sizeof (int)) < 0) {
<         Error(I18N(25, "Could not write metadata cache file
name to incrementals cache.));
<         return (-1);
<     }
<     DebugPD(ASCII("MDID: %s LEN: %d"), mdId, len);
<     if (write(incrCacheFd, mdId, len) < 0) {
<         Error(I18N(25, "Could not write metadata cache file
name to incrementals cache.));
<         return (-1);
<     }
---
>
>     DebugPD(ASCII("incr cache file  fd: %d"),
incrCacheFd);
>
>     len = strlen(mdId) + 1;
>     if (write(incrCacheFd, &len, sizeof (int)) < 0)
{
>         Error(I18N(25, "Could not write metadata
cache file name to incrementals cache.));
>         return (-1);
>     }
>     DebugPD(ASCII("MDID: %s LEN: %d"), mdId, len);
>     if (write(incrCacheFd, mdId, len) < 0) {
>         Error(I18N(25, "Could not write metadata
cache file name to incrementals cache.));
>         return (-1);
>     }
>     if (newRestoreDesign == 1) {
>         len = strlen(mdInodeId) + 1;
>         if (write(incrCacheFd, &len, sizeof
(int)) < 0) {

```

```

>                                     Error(I18N(25, "Could not write
metadata cache file name to incrementals cache."));
>                                     return (-1);
>                                     }
>                                     DebugPD(ASCII("mdInodeId: %s LEN: %d"),
mdInodeId, len);
>                                     if (write(incrCacheFd, mdInodeId, len) <
0) {
>                                     Error(I18N(25, "Could not write
metadata cache file name to incrementals cache."));
>                                     return (-1);
>                                     }
>                                     len = strlen(InodeIndexId) + 1;
>                                     if (write(incrCacheFd, &len, sizeof
(int)) < 0) {
>                                     Error(I18N(25, "Could not write
Inode Index file name to incrementals cache."));
>                                     return (-1);
>                                     }
>                                     if (write(incrCacheFd, InodeIndexId,
len) < 0) {
>                                     Error(I18N(25, "Could not write
metadata cache file name to incrementals cache."));
>                                     return (-1);
>                                     }
>                                     }
>
1041,1052c1040,1051
<         if ((incrCacheFd = open(incrCacheFile, O_RDONLY |
O_BINARY)) < 0) {
<             DebugPD(ASCII("Could not open cache file for reading
incrementals information."));
<             createFlag = DM_TRUE;
<         } else {
<             incrRestores = DM_TRUE;
<             createFlag = DM_FALSE;
<             if (read(incrCacheFd, &len, sizeof (int)) < sizeof
(int)) {
<                 DebugPD(ASCII("Error reading incrementals file:
%s"), ErrorMessage(errno));
<                 close(incrCacheFd);
<                 unlink(incrCacheFile);
<                 return (DM_ERROR);
<             }
---
>         if ((incrCacheFd = open(incrCacheFile, O_RDONLY
| O_BINARY)) < 0) {

```

```

>             DebugPD(ASCII("Could not open cache file
for reading incrementals information.));
>             createFlag = DM_TRUE;
>         } else {
>             incrRestores = DM_TRUE;
>             createFlag = DM_FALSE;
>             if (read(incrCacheFd, &len, sizeof
(int)) < sizeof (int)) {
>                 DebugPD(ASCII("Error reading
incrementals file: %s"), ErrorMessage(errno));
>                 close(incrCacheFd);
>                 unlink(incrCacheFile);
>                 return (DM_ERROR);
>             }
1054c1053
<             mdId = (char *) Malloc(len);
---
>             mdId = (char *) Malloc(len);
1056,1071c1055,1059
<             if (read(incrCacheFd, mdId, len) < len) {
<                 DebugPD(ASCII("Error reading incrementals file:
%s"), ErrorMessage(errno));
<                 close(incrCacheFd);
<                 unlink(incrCacheFile);
<                 return (DM_ERROR);
<             }
<
<             DebugPD(ASCII("MDID: %s LEN: %d"), mdId, len);
< #ifdef CRM_ENABLED
<             if (useCrmID == DM_TRUE) {
<                 IDType mdIdNum = atol(mdId);
<                 if (mdFile >= 0) {
<                     rcf->crmf_close(mdFile);
<                     if ((mdFile = rcf->crmf_open(mdIdNum,
CRMF_READ)) < 0) {
<                         Error(I18N(106, "Could not open cache
file for reading incrementals metadata information.));
<                         return (DM_ERROR);
---
>                     if (read(incrCacheFd, mdId, len) < len) {
>                         DebugPD(ASCII("Error reading
incrementals file: %s"), ErrorMessage(errno));
>                         close(incrCacheFd);
>                         unlink(incrCacheFile);
>                         return (DM_ERROR);
1072a1061,1119

```



```

>             DebugPD(ASCII("MDID: %s LEN: %d"), mdId,
len);
>             if (newRestoreDesign == 1) {
>                 if (read(incrCacheFd, &len,
sizeof (int)) < sizeof (int)) {
>                     DebugPD(ASCII("Error
reading incrementals file: %s"), ErrorMessage(errno));
>                     close(incrCacheFd);
>                     unlink(incrCacheFile);
>                     return (DM_ERROR);
>                 }
>                 mdInodeId = (char *)
Malloc(len);
>                 if (read(incrCacheFd, mdInodeId,
len) < len) {
>                     DebugPD(ASCII("Error
reading incrementals file: %s"), ErrorMessage(errno));
>                     close(incrCacheFd);
>                     unlink(incrCacheFile);
>                     return (DM_ERROR);
>                 }
>                 DebugPD(ASCII("mdInodeId: %s
LEN: %d"), mdInodeId, len);
>                 if (read(incrCacheFd, &len,
sizeof (int)) < sizeof (int)) {
>                     DebugPD(ASCII("Error
reading incrementals file: %s"), ErrorMessage(errno));
>                     close(incrCacheFd);
>                     unlink(incrCacheFile);
>                     return (DM_ERROR);
>                 }
>                 InodeIndexId = (char *)
Malloc(len);
>                 if (read(incrCacheFd,
InodeIndexId, len) < len) {
>                     DebugPD(ASCII("Error
reading incrementals file: %s"), ErrorMessage(errno));
>                     close(incrCacheFd);
>                     unlink(incrCacheFile);
>                     return (DM_ERROR);
>                 }
>                 DebugPD(ASCII("InodeIndexId: %s
LEN: %d"), InodeIndexId, len);
>             }
>             if (useMdID == DM_TRUE) {
>                 if (mdFile >= 0) {
>                     close(mdFile);

```

```

>                                     sprintf(mdfname,
"%s/mdcache/%s", AppHome, mdId);
>                                     if ((mdFile =
open(mdfname, O_RDONLY | O_BINARY)) < 0) {
>                                     Error(I18N(106,
"Could not open cache file for reading incrementals metadata
informatio
n."));
>                                     return
(DM_ERROR);
>                                     }
>                                     }
>                                     if (newRestoreDesign == 1) {
>                                     if (inodeIndexFile >= 0)
{
>                                     }
close(inodeIndexFile);
>                                     sprintf(inodeIndexfname, "%s/mdcache/%s", AppHome,
InodeIndexId);
>                                     if
((inodeIndexFile = open(inodeIndexfname, O_RDONLY | O_BINARY)) <
0) {
>                                     Error(I18N(106, "Could not open cache file for reading
incrementals metadata in
formation."));
>                                     return
(DM_ERROR);
>                                     }
>                                     }
>                                     if (MdInodeFile >= 0) {
>                                     }
close(MdInodeFile);
>                                     sprintf(mdInodeFileName, "%s/mdcache/%s", AppHome, mdInodeId);
>                                     if ((MdInodeFile
= open(mdInodeFileName, O_RDONLY | O_BINARY)) < 0) {
>                                     Error(I18N(106, "Could not open cache file for reading
incrementals metadata in
formation."));
>                                     return
(DM_ERROR);
>                                     }
>                                     }
>                                     }

```

```

>                                     }
1074,1086d1120
<                                     } else
< #endif
<         if (useMdID == DM_TRUE) {
<             if (mdFile >= 0) {
<                 close(mdFile);
<                 sprintf(mdfname, "%s/mdcache/%s", AppHome,
mdId);
<                 if ((mdFile = open(mdfname, O_RDONLY |
O_BINARY)) < 0) {
<                     Error(I18N(106, "Could not open cache file
for reading incrementals metadata information."));
<                     return (DM_ERROR);
<                 }
<             }
<         }
<     }
1091,1096c1125,1142
<     Log(I18N(107, "Get file information from cache."));
<     mdImage = new MDImage(MD_TAPEBUFSIZE, mdFile);
< #ifdef CRM_ENABLED
<     mdImage->setCrmFaces(rcf);
<     mdImage->setUseCrmFsysFlag(useCrmID);
< #endif
---
>         Log(I18N(107, "Get file information from
cache."));
>         if (newRestoreDesign == 1) {
>             long mdSize = 61440;
>             long fileMdSize = 61440;
>             DebugPD(ASCII("metadataSize = %ld,
mdFile=%d\n"), mdSize, mdFile);
>             mdImage = new NewMDImage(mdSize,
mdFile);
>             if (mdInodeId == NULL) {
>                 Error(I18N(-1, "No MD Inode
Index File"));
>                 goto done;
>             } else {
>                 DebugUI("Md Inode cache filename
=%s", mdInodeFileName);
>                 DebugPD(ASCII("file metadataSize = %ld,
mdFile=%d\n"), fileMdSize, mdFile);
>                 fileMdImage = new NewMDImage(fileMdSize,
MdInodeFile);
>             }

```

```

>          } else {
>              mdImage = new MDImage(MD_TAPEBUFSIZE,
mdFile);
>          }
>
1228,1243d1273
< #ifndef CRM_ENABLED
<     if (rcf != NULL && useCrmID == DM_TRUE) {
<         while ((read_count = rcf->crmf_read(mapFile,
mapbuf, mapSize)) == mapSize) {
<             if ((mapData = (MapExtent *)
realloc(mapData, curSize + mapSize)) == NULL) {
<                 Error(I18N(31, "Cannot reallocate
memory"));
<                 return (-1);
<             }
<             memcpy((mapData + curSize), mapbuf,
mapSize);
<             curSize += mapSize;
<             memset(mapbuf, 0, mapSize);
<             read_count = 0;
<         }
<     } else {          // useMdID == DM_TRUE
<
< #endif
1255,1257d1284
< #ifndef CRM_ENABLED
<     }
< #endif
1317,1324d1343
< #ifndef CRM_ENABLED
<     if (useCrmID == DM_TRUE) {
<         if (mdFile != -1)
<             rcf->crmf_close(mdFile);
<         if (mapFile != -1)
<             rcf->crmf_close(mapFile);
<     } else
< #endif
1353,1355d1371
< #ifndef CRM_ENABLED
<     if (rcf == NULL) {
< #endif
1360,1368d1375
< #ifndef CRM_ENABLED
<     } else {
<         IDType mdn = rcf->crmf_close(mdSaveFd);

```

```

<          if (mdn < 0) {
<              Error(I18N(35, "error in closing metadata save
file."));
<              return (-1);
<          }
<      }
< #endif
1369a1377,1380
>      if (inodeIndexTable != NULL) {
>          free (inodeIndexTable);
>          inodeIndexTable = NULL;
>      }
1550c1561
<
---
> Log(I18N(3,"Restoring IMAGE - Used Block Only"));
1889,1891c1900,1908
<      ++endCount;
<      break;
<
---
>          if (newRestoreDesign == 0) {
>              ++endCount;
>              break;
>          }
>      case DIR_METADATA_HEADER:
>          if (newRestoreDesign == 1) {
>              ++endCount;
>              break;
>          }
2155a2173,2453
>
> void* getChildInfoList(ino_t inode_num, __int64 offset,
DirChildInfoList *dirChildList, InodeCell **dirInfo)
> {
>     char *dirBuffer = NULL;
>     unsigned long dirBufferSize =0;
>     char *buff = NULL;
>     MDInodeRec *mdInodep = new MDInodeRec();
>     struct CelestraTapeDirInfo {
>         long inoNum;
>         short recLen;          /* To be filled by bigImpl.
*/
>         short nameLen;
>         char name[1];          /* Null terminated and
variable length */
>     };

```

```

>     struct CelestraTapeDirInfo *currEntry;
>     DirChildInfoList *currChild, *prevChild, *tmpDirChildList,
>     *tmpDirChildList1;
>     int sizeProcessed=0;
>     int childCount=0;
>     SIDFField field;
>     GenericList *attribListp = (struct GenericList *)
NewGenList(5);
>    _ATTRIBStreamInfo *attribCell;
>     int fidNum;
>     u_int buffSize;
> #ifdef DM_WINDOWS_NT
>     u_long dirSize = 0;
>     void *pluginSpecificData = NULL;
>     u_long pluginSpecificDataSize = 0;
>
> #endif
>
>
>     mdImage->seek(offset, SEEK_SET);
>     field.read(*mdImage);
>     if ((fidNum = field.getFidNumber()) != METADATA_INODE) {
>         DebugPD(ASCII("Unexpected FID: %d, while expecting
inode record."), fidNum);
>         Error(I18N(-1, "Invalid Fid for Metadata Record "));
>         return(NULL);
>     }
>     field.getData(buff, buffSize);
>     if (buff[0] & STANDARD_ATTRIBUTES_PRESENT) {
>         DebugPD(ASCII("Standard attributes present.));
>         memcpy(mdInodep, buff + 1, sizeof(MDInodeRec));
>         if (mdInodep->inoNum != inode_num) {
>             Error(I18N(-1, "Inodes mismatch in Inode Index"));
>             return(NULL);
>         }
>         DebugPD(ASCII(" mdinonum: %u links : %d size : %llu
mode : %x datasizetofollow: %d"),
>             mdInodep->inoNum, mdInodep->nlink,
GET_SIZE(mdInodep->size), mdInodep->mode, mdInodep->dataSzToFollow));
>
>         /* need to fill in the plugin specific data */
>
>         if (buff[0] & PRIMARY_ATTRIBUTE_ONLY) {
>             attribCell = (_ATTRStreamInfo *)
MALLOC(sizeof(_ATTRStreamInfo));
>             attribCell->type = MD_PRIMARY_ATTRIB;

```

```

>         attribCell->streamSize = mdInodep-
>dataSzToFollow;
>         attribCell->streamSize = (attribCell-
>streamSize + 3) & (~3);
>         attribCell->streamData = malloc(attribCell-
>streamSize);
>         if (attribCell->streamData == NULL) {
>             Error(I18N(63, "Could
not allocate memory for attribute stream data."));
>             return(NULL);
>         }
>
>         DebugPD(ASCII("streamsize =%d\n"), attribCell-
>streamSize);
>         if (mdImage->read((char *)
attribCell->streamData, attribCell->streamSize) < attribCell-
>streamSize) {
>             Error(I18N(64, "Error reading
metadata."));
>             return(NULL);
>         }
>         AddToGenList(attribListp,
attribCell);
>     }
>     } else {
>         memset(mdInodep, 0, sizeof(MDInodeRec));
>     }
>
>     field.read(*mdImage);
>     while ((fidNum = field.getFidNumber()) ==
ATTRIBUTE_STREAM_HEADER) {
>         attribCell = (AttribStreamInfo *)
Malloc(sizeof(AttribStreamInfo));
>         attribCell->type = (char) field;
>
>         /* handle following fields based on the attribute
type */
>         /* Nothing defined currently. */
>         field.read(*mdImage);
>         while ((fidNum = field.getFidNumber()) !=
ATTRIBUTE_STREAM_DATA) {
>             switch (attribCell->type &
ATTRIB_TYPE_MASK) {
>                 case DUMMY_ATTRIB:
>                     DebugPD(ASCII(" DUMMY attrib
found."));
>                     break;

```

```

>
>
>         }
>         field.read(*mdImage);
>         fidNum = field.getFidNumber();
>
>     }
>     /* read attribute stream data. */
>     attribCell->streamSize = (long) field;
>     attribCell->streamSize = (attribCell->streamSize +
3) & (~3);
>     attribCell->streamData = malloc(attribCell-
>streamSize);
>     if (attribCell->streamData == NULL) {
>         Error(I18N(63, "Could not allocate
memory for attribute stream data."));
>         return NULL;
>     }
>     if (mdImage->read((char *) attribCell-
>streamData, attribCell->streamSize) < attribCell->streamSize) {
>         Error(I18N(64, "Error reading
metadata."));
>         return NULL;
>     }
>     AddToGenList(attribListp, attribCell);
>
>     /* Look for next attribute stream */
>     field.read(*mdImage);
>     fidNum = field.getFidNumber();
>
> }
> if (((*dirInfo) = (InodeCell *) malloc
(sizeof(InodeCell))) == NULL) {
>     Error(I18N(-1, "malloc failed"));
>     return(NULL);
> }
> (*dirInfo)->mode = mdInodep->mode;
> (*dirInfo)->size = mdInodep->size;
> (*dirInfo)->uid = mdInodep->uid;
> (*dirInfo)->gid = mdInodep->gid;
> (*dirInfo)->nlink = mdInodep->nlink;
> (*dirInfo)->atime = mdInodep->atime;
> (*dirInfo)->mtime = mdInodep->mtime;
>
> for (attribCell = (AttribStreamInfo *)
GetFirstGenericList(attribListp);
>     attribCell != NULL;
>     attribCell = (AttribStreamInfo *)
GetNextGenericList(attribListp)) {
>     switch (attribCell->type & ATTRIB_TYPE_MASK) {

```



```

>         case DUMMY_ATTRIB:
>             break;
>
>         case MD_PRIMARY_ATTRIB:
>         case MD_DIR_ATTRIB:
>             dirBuffer = (char*)attribCell-
>streamData;
>             dirBufferSize = attribCell->streamSize;
>             break;
>         case MD_PSI_ATTRIB:
>             (*dirInfo)->pluginSpecificData =
attribCell->streamData;
>             (*dirInfo)->pluginSpecificDataSize =
attribCell->streamSize;
>             break;
>         default:
>             break;
>     }
>     Free(attribCell);
> }
>     currEntryp = (CelestraTapeDirInfo *)dirBuffer;
>     DebugPD(ASCII("currEntryp = %d\n"), currEntryp);
>     if (currEntryp == NULL) {
>         return (NULL);
>     }
>     prevChild = NULL;
>     DebugPD(ASCII("sizeProcessed=%d, mdInodep-
>dataSzToFollow=%d\n"), sizeProcessed, mdInodep-
>dataSzToFollow);
>     while (sizeProcessed < dirBufferSize) {
>         childCount++;
>         if ((currChild = (DirChildInfoList*) malloc
(sizeof (struct DirChildInfoList))) == NULL) {
>             Error(I18N(-1, "malloc failed."));
>             return(NULL);
>         }
>         currChild->fName = strdup(currEntryp->name);
>         currChild->inodNo = currEntryp->inoNum;
>         currChild->nextElement = NULL;
>         if (prevChild != NULL) {
>             prevChild->nextElement = currChild;
>         } else {
>             tmpDirChildList =currChild;
>         }
>         DebugPD(ASCII("child 1:name=%s,inode=%d,
reclen= %d\n"), currEntryp->name, currEntryp->inoNum,
currEntryp->recL

```

```

en);
>
>         sizeProcessed+=currEntry->recLen;
>         currEntry = (CelestraTapeDirInfo *)
(void *) (((char *) (void *) currEntry) + currEntry->recLen);
>         prevChild = currChild;
>     }
>     tmpDirChildList1 = tmpDirChildList;
>     DebugPD(ASCII("Verifying the Children Linked List\n"));
>     while(tmpDirChildList != NULL) {
>         DebugPD(ASCII(" Child InodeName : %s is a child
returned by "), tmpDirChildList->fName);
>         tmpDirChildList = tmpDirChildList->nextElement;
>     }
>     return (tmpDirChildList1);
>
> }
> int loadInodeIndexTable ()
> {
>     DebugPD(ASCII("inside loadInodeIndexTable:
inodeIndexFile=%d\n"), inodeIndexFile);
>     inodeIndexTable = (InodeIndexRec *)
malloc((maxInodeNumber+1) * sizeof(InodeIndexRec));
>     if (inodeIndexTable == NULL) {
>         Error(I18N(-1, "malloc failed."));
>         return (-1);
>     }
>     return (read(inodeIndexFile, inodeIndexTable,
(maxInodeNumber+1)*sizeof(InodeIndexRec)) );
> }
>
> void getInodeMetaDataInfo(int inode_num, MDInodeRec *mdInodep,
struct GenericList *attribStreamListp)
> {
>     __int64 offset;
>     char *buff;
>    _ATTRIBStreamInfo *attribCell;
>     int fidNum;
>     SIDFField field;
>     u_int buffSize;
>
>     DebugPD(ASCII("Getting InodeMetaDataInfo for ino=%d\n"),
inode_num);
>
>     offset = inodeIndexTable[inode_num].typeAndOffset;
>
>     DebugPD(ASCII("offset in metadata file= %d\n"), offset);
>     fileMdImage->seek(offset, SEEK_SET);

```

```

>     field.read(*fileMdImage);
>     if ((fidNum = field.getFidNumber()) != METADATA_INODE) {
>         DebugPD(ASCII("Unexpected FID: %d, while expecting
inode record."), fidNum);
>         Error(I18N(-1, "Invalid Fid for Metadata Record "));
>         return;
>     }
>     field.getData(buff, buffSize);
>     if (buff[0] & STANDARD_ATTRIBUTES_PRESENT) {
>         DebugPD(ASCII("Standard attributes present."));
>         memcpy(mdInodep, buff + 1, sizeof(MDInodeRec));
>         if (mdInodep->inoNum != inode_num) {
>             Error(I18N(-1, "Inodes mismatch in Inode Index"));
>             return;
>         }
>         if (buff[0] & PRIMARY_ATTRIBUTE_ONLY) {
>             DebugPD(ASCII("Primary attributes present."));
>             attribCell = (AttribStreamInfo *)
Malloc(sizeof(AttribStreamInfo));
>             attribCell->type = MD_PRIMARY_ATTRIB;
>             attribCell->streamSize = mdInodep-
>dataSzToFollow;
>             attribCell->streamSize = (attribCell-
>streamSize + 3) & (~3);
>             DebugPD(ASCII("streamsize
=%d\n"), attribCell->streamSize);
>             attribCell->streamData =
malloc(attribCell->streamSize);
>             if (attribCell->streamData ==
NULL) {
>                 Error(I18N(63,
"Could not allocate memory for attribute stream data."));
>                 return;
>             }
>             if (fileMdImage->read((char *) attribCell-
>streamData, attribCell->streamSize)
>                 < attribCell->streamSize) {
>                 Error(I18N(64, "Error reading
metadata."));
>                 return;
>             }
>             AddToGenList(attribStreamListp,
attribCell);
>         }
>     } else {
>         memset(mdInodep, 0, sizeof(MDInodeRec));
>     }

```

```

>
>     field.read(*fileMdImage);
>     while ((fidNum = field.getFidNumber()) ==
ATTRIBUTE_STREAM_HEADER) {
>         attribCell = (AttribStreamInfo *)
Malloc(sizeof(AttribStreamInfo));
>         attribCell->type = (char) field;
>
>         /* handle following fields based on the attribute
type */
>         /* Nothing defined currently. */
>         field.read(*fileMdImage);
>         while ((fidNum = field.getFidNumber()) !=
ATTRIBUTE_STREAM_DATA) {
>             switch (attribCell->type &
ATTRIB_TYPE_MASK) {
>                 case DUMMY_ATTRIB:
>                     DebugPD(ASCII(" DUMMY attrib
found."));
>                     break;
>
>             }
>             field.read(*fileMdImage);
>             fidNum = field.getFidNumber();
>         }
>
>         /* read attribute stream data. */
>         attribCell->streamSize = (long) field;
>         attribCell->streamSize = (attribCell->streamSize +
3) & (~3);
>         attribCell->streamData = malloc(attribCell->
streamSize);
>         if (attribCell->streamData == NULL) {
>             Error(I18N(63, "Could not allocate
memory for attribute stream data."));
>             return;
>         }
>         if (fileMdImage->read((char *) attribCell->
streamData, attribCell->streamSize) < attribCell->streamSize) {
>             Error(I18N(64, "Error reading
metadata."));
>             return;
>         }
>         AddToGenList(attribStreamListp, attribCell);
>
>         /* Look for next attribute stream */
>         field.read(*fileMdImage);

```

```
>         fidNum = field.getFidNumber();
>     }
>
>     if (buff != NULL) {
>         //     free(buff);
>     }
> }
```

Exhibit K – Differences between versions 1.1.2.2 and 1.1.2.3 of rip.hpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.2 -r 1.1.2.3  rip.hpp
Index: rip.hpp
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/rip.hpp,v
retrieving revision 1.1.2.2
retrieving revision 1.1.2.3
diff -r1.1.2.2 -r1.1.2.3
1c1
< /* $Id: rip.hpp,v 1.1.2.2 2001/03/16 09:54:02 anju Exp $
Copyright (c) 2001, Legato Systems, Inc. */
---
> /* $Id: rip.hpp,v 1.1.2.3 2002/10/19 22:29:02 nsq Exp $
Copyright (c) 2002, Legato Systems, Inc. */
67a68,71
> extern int newRestoreDesign;
> extern struct InodeIndexRec *inodeIndexTable;
>
>
90a95,97
> extern void* getChildInfoList(ino_t inode_num, __int64 offset,
DirChildInfoList *dirChildList, InodeCell **dirInfo);
> extern void getInodeMetaDataInfo(int inode_num, MDInodeRec
*mdInodep, struct GenericList *attribStreamListp);
> extern int loadInodeIndexTable();
```

Exhibit L – Differences between versions 1.1.2.10 and 1.1.2.11 of rtrv_filemd.cpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.10 -r 1.1.2.11 rtrv_filemd.cpp
Index: rtrv_filemd.cpp
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/rtrv_filemd.cpp,v
retrieving revision 1.1.2.10
retrieving revision 1.1.2.11
diff -r1.1.2.10 -r1.1.2.11
2,11c2
< #ident "$Id: rtrv_filemd.cpp,v 1.1.2.10 2001/07/27 07:21:50
harish Exp $ Copyright (c) 2001, Legato Systems, Inc."
< #endif
<
< /*
<  * Copyright (c) 2001, Legato Systems, Inc.
<  *
<  * All rights reserved.
<  */
< #if !defined(lint) && !defined(SABER)
< static char rcsid[] = "@(#) $Id: rtrv_filemd.cpp,v 1.1.2.10
2001/07/27 07:21:50 harish Exp $ " DM_BUILD;
---
> #ident "$Id: rtrv_filemd.cpp,v 1.1.2.11 2002/10/19 22:31:12
nsq Exp $ Copyright (c) 2002, Legato Systems, Inc."
16d6
<  * Copyright (c) 2000, Legato Systems Incorporated.
18a9,11
>  * Revision 1.1.2.11 2002/10/19 22:31:12 nsq
>  * LGTpa45351: added code to use indexing of metadata for FBF
retrievals
>  *
217c210
<
---
> #include "rip.hpp"
261c254
< dm_status getNextStat;
---
> dm_status getNextStat = DM_OK;
310,327c303,318
```



```

>                                     }
>                                     if (attribCell-
>streamData != NULL) {
>                                     free(attribCell-
>streamData);
>                                     }
>                                     free(attribCell);
329c320,321
<         }
---
>         ResetGenList(attribListp);
> #endif
330a323,346
>         while (fileCellp != NULL) {
>             getInodeMetaDataSource(fileCellp->inoNum,
&mdInode,attribListp);
>
>
////////////////////////////////////////
>             // while ((getNextStat =
GetNextMetaDataSource(MD_NEXTREC_FILE,
>             // &mdInode, attribListp)) == DM_OK)
>
>             pluginSpecificData = NULL;
>             pluginSpecificDataSource = 0;
>
>             for (attribCell = (AttribStreamInfo *)
GetFirstGenericList(attribListp); attribCell != NULL; attribCell
= (Attr
ibStreamInfo *) GetNextGenericList(attribListp)) {
>                 switch (attribCell->type &
ATTRIB_TYPE_MASK)
>                 {
>                     case MD_PRIMARY_ATTRIB:
>                         fileData = (char
*) attribCell->streamData;
>                         break;
>                     case MD_PSI_ATTRIB:
>
pluginSpecificData = attribCell->streamData;
>
pluginSpecificDataSource = attribCell->streamSize;
>                         break;
>                     default:
>                         break;
>                 }
>             }

```

```

332,348d347
<     DebugPD(ASCII("inode %d nlinks %d  mode %d uid %d  gid
%d  fileSz %lld dataSz %d \n"),
<                                     mdInode.inoNum,
<                                     mdInode.nlink,
<                                     mdInode.mode,
<                                     mdInode.uid,
<                                     mdInode.gid,
<                                     GET_SIZE(mdInode.size),
<                                     mdInode.dataSzToFollow);
<
<     if ((mdInode.inoNum > maxInodeNumber) || (mdInode.mode
== 0)) {
<         Error(I18N(66, "Corrupted Metadata stream"));
<         DebugPD(ASCII("current inode number : %u "),
currInoNum);
<         retVal = DM_ERROR;
<         goto done;
<     }
<
<     currInoNum = mdInode.inoNum;
350,358c349,375
<     /* skip files in requested list if metadata record is
not available */
<     /* Ideally it should not occur for consistent file
systems. */
<     while ((fileCellp != NULL) && (fileCellp->inoNum <
mdInode.inoNum)) {
<         GetFileName(fileCellp->parentp,
GetNodeName(fileCellp),fileName);
<         DebugPD(ASCII("Metadata does not contain any
information for file : %s"),fileName);
<         DebugPD(ASCII("The File system does not appear to be
in consistent state"));
<         Free(fileCellp);
<         fileCellp = (FileInfoCell *)
GetNextGenericList(fileTable);
<     }
---
>
>         DebugPD(ASCII("inode %d nlinks %d  mode
%d uid %d  gid %d  fileSz %lld dataSz %d \n"),
>
mdInode.inoNum,
>
mdInode.nlink,
>
mdInode.mode,

```

```

>
mdInode.uid,
>
mdInode.gid,
>
GET_SIZE(mdInode.size),
>
mdInode.dataSzToFollow);
>
>
if ((mdInode.inoNum > maxInodeNumber) ||
(mdInode.mode == 0)) {
>
Error(I18N(66, "Corrupted
Metadata stream"));
>
DebugPD(ASCII("current inode
number : %u "), currInoNum);
>
retVal = DM_ERROR;
>
goto done;
>
}
>
>
currInoNum = mdInode.inoNum;
>
>
/* skip files in requested list if
metadata record is not available */
>
/* Ideally it should not occur for
consistent file systems. */
>
while ((fileCellp != NULL) &&
(fileCellp->inoNum < mdInode.inoNum)) {
>
GetFileName(fileCellp->parentp,
GetNodeName(fileCellp), fileName);
>
DebugPD(ASCII("Metadata does not
contain any information for file : %s"), fileName);
>
DebugPD(ASCII("The File system
does not appear to be in consistent state"));
>
Free(fileCellp);
>
fileCellp = (FileInfoCell *)
GetNextGenericList(fileTable);
>
}
360,363c377,380
<
if(mdInode.inoNum == SECURITY_FILE_ID) {
<
DebugPD(ASCII("initializing security file Info
cell"));
<
initializeSecurityFileInfoCell((&mdInode), ((char*)pluginSpecific
Data), (pluginSpecificDataSize));
<
}
---
>
if(mdInode.inoNum == SECURITY_FILE_ID) {

```

```

>
DebugPD(ASCII("initializing security file Info cell"));
>
initializeSecurityFileInfoCell((&mdInode), ((char*)pluginSpecific
Data), (pluginSpecificDataSize))
;
>
    }
365,369c382,386
<     if (fileCellp != NULL) {
<         if (fileCellp->inoNum == mdInode.inoNum) {
<             delFlag = DO_NOT_DELETE;
<             inodeInfop = (InodeCell *) Malloc(sizeof
(InodeCell));
<             GetInodeInfo(&mdInode, inodeInfop);
---
>             if (fileCellp != NULL) {
>                 if (fileCellp->inoNum ==
mdInode.inoNum) {
>                     delFlag = DO_NOT_DELETE;
>                     inodeInfop = (InodeCell
*) Malloc(sizeof (InodeCell));
>                     GetInodeInfo(&mdInode,
inodeInfop);
371,373c388,390
<
allocateAndAssignPSIDataToInodeInfo(inodeInfop,
<
(char*)pluginSpecificData,
<
pluginSpecificDataSize);
---
>
allocateAndAssignPSIDataToInodeInfo(inodeInfop,
>
(char*)pluginSpecificData,
>
pluginSpecificDataSize);
375,376c392,393
<
<
fileCellp->inodeInfop = inodeInfop;
---
>
>
fileCellp->inodeInfop =
inodeInfop;
378c395
<
GetFileName(fileCellp->parentp,
GetNodeName(fileCellp), fileName);

```

```

---
>
>parentp, GetNodeName(fileCellp, fileName);
380,434c397,451
<
<
/* take action based on file type */
switch (GET_INODE_MODE(inodeInfop->mode)) {
<
<
case S_IFIFO: /* defined as
invalid on nt */
<
<
DebugPD(ASCII(" fifo
"));
<
<
if (createFlag ==
DM_TRUE) {
<
<
if
(mkfifo(fileName, inodeInfop->mode) == -1)
<
DebugUI(ASCII("Cannot create fifo %s: %s"),fileName,
ERROR_MESSAGE);
<
<
else
<
AnnounceDone(fileName);
<
}
<
delFlag =
FILE_DONE_DELETE;
<
break;
<
case S_IFSOCK: /* defined as
invalid on nt */
<
DebugPD(ASCII(" socket
"));
<
if (createFlag ==
DM_TRUE) {
<
AnnounceDone(fileName);
<
}
<
delFlag = ERROR_DELETE;
<
/* nothing to do for
sockets */
<
break;
<
case S_IFBLK:
<
case S_IFCHR:
<
if (createFlag ==
DM_TRUE) {
<
memcpy(&devMajorNum, fileData, sizeof (u_long));
<
memcpy(&devMinorNum, fileData + sizeof (u_long), sizeof
(u_long));

```

```

<
DebugPD(ASCII("Device file MajorNo = %u, MinorNo = %u"),
devMajorNum, devMinorNum);
<
devNum =
makedev(devMajorNum, devMinorNum);
<
if
(mknod(fileName, inodeInfop->mode, devNum) == -1)
<
Log(I18N(96, "Cannot create block/char special device %s: %s."),
<
fileName, ERROR_MESSAGE);
<
else
<
AnnounceDone(fileName);
<
}
<
delFlag =
FILE_DONE_DELETE;
<
break;
<
case S_IFLNK: /* defined as
invalid on nt */
DebugPD(ASCII(" symlink
"));
<
if (createFlag ==
DM_TRUE) {
memcpy(linkName,
fileData, (int) GET_SIZE(inodeInfop->size));
<
linkName[GET_SIZE(inodeInfop->size)] = '\0';
<
MakeSymLink(linkName, fileName);
<
}
<
delFlag =
FILE_DONE_DELETE;
<
break;
<
case S_IFREG:
DebugPD(ASCII(" regular
file "));
<
if
(IS_FILE_ENCRYPTED(fileCellp)) {
if (createFlag
== DM_TRUE) {
AnnounceDone(fileName);
<
}
<
DebugPD(ASCII("%s is an encrypted file, will not be
restored."), fileName);

```

```

<                                     Warning(I18N(-1,
"%s IS AN ENCRYPTED FILE, NOT RESTORED. ENCRYPTED FILE NOT
SUPPORTED.")
,fileName);
<                                     delFlag =
ERROR_DELETE;
<                                     break;
<                                     }
---
>                                     /* take action based on
file type */
>                                     switch
(GET_INODE_MODE(inodeInfop->mode)) {
>                                     case S_IFIFO: /*
defined as invalid on nt */
>                                     if
(createFlag == DM_TRUE) {
>                                     if (mkfifo(fileName, inodeInfop->mode) == -1)
>                                     DebugUI(ASCII("Cannot create fifo %s: %s"),fileName, ERROR_MESS
AGE);
>                                     else
>                                     AnnounceDone(fileName);
>                                     }
>                                     delFlag
= FILE_DONE_DELETE;
>                                     break;
>                                     case S_IFSOCK:
/* defined as invalid on nt */
>                                     if
(createFlag == DM_TRUE) {
>                                     AnnounceDone(fileName);
>                                     }
>                                     delFlag
= ERROR_DELETE;
>                                     /*
nothing to do for sockets */
>                                     break;
>                                     case S_IFBLK:

```

```

>
>                                     case S_IFCHR:
>                                     if
(createFlag == DM_TRUE) {
>
memcpy(&devMajorNum, fileData, sizeof (u_long));
>
memcpy(&devMinorNum, fileData + sizeof (u_long), sizeof
(u_long));
>
DebugPD(ASCII("Device file MajorNo = %u, MinorNo = %u"),
devMajorNum, d
evMinorNum);
>
devNum = makedev(devMajorNum, devMinorNum);
>
if (mknod(fileName, inodeInfop->mode, devNum) == -1)
>
Log(I18N(96, "Cannot create block/char special device %s: %s.")
,
>
fileName, ERROR_MESSAGE);
>
else
>
AnnounceDone(fileName);
>
>                                     }
>                                     delFlag
= FILE_DONE_DELETE;
>                                     break;
>                                     case S_IFLNK: /*
defined as invalid on nt */
>
DebugPD(ASCII(" symlink "));
>                                     if
(createFlag == DM_TRUE) {
>
memcpy(linkName, fileData, (int) GET_SIZE(inodeInfop->size));
>
linkName[GET_SIZE(inodeInfop->size)] = '\0';
>
MakeSymLink(linkName, fileName);
>
>                                     }
>                                     delFlag
= FILE_DONE_DELETE;
>                                     break;
>                                     case S_IFREG:

```



```

>
DebugPD(ASCII(" regular file "));
>
(IS_FILE_ENCRYPTED(fileCellp)) {
>
if (createFlag == DM_TRUE) {
>
AnnounceDone(fileName);
>
}
>
DebugPD(ASCII("%s is an encrypted file, will not be
restored."),fileNam
e);
>
Warning(I18N(-1, "%s IS AN ENCRYPTED FILE, NOT
RESTORED.ENCRYPTED FILE
NOT SUPPORTED."),fileName);
>
delFlag = ERROR_DELETE;
>
break;
>
}
436,444c453,461
<
if (createFlag ==
DM_TRUE) {
<
fd =
IGScreat64(fileName, FILE_CREATION_FLAGS);
<
} else {
<
fd =
IGSopen64(fileName,FILE_WRITE_FLAGS);
<
}
<
if (fd ==
INVALID_HANDLE) {
<
DebugPD(ASCII("Cannot create file %s"), fileName);
<
err = 1;
---
>
if
(createFlag == DM_TRUE) {
>
fd = IGScreat64(fileName, FILE_CREATION_FLAGS);
>
} else {
>
fd = IGSopen64(fileName,FILE_WRITE_FLAGS);
>
}

```

```

>
>                                     if (fd
== INVALID_HANDLE) {
>
DebugPD(ASCII("Cannot create file %s"), fileName);
>
err = 1;
446,448c463,465
<                                     /* this function
will return 1 if fileName is not registry file */
<                                     /* else it will
return 0 */
<                                     err =
checkAndMarkRegistryFileInodeCell(inodeInfop,fileName);
---
>
/* this function will return 1 if fileName is not registry file
*/
>
/* else it will return 0 */
>
err = checkAndMarkRegistryFileInodeCell(inodeInfop,fileName);
450,461c467,483
<                                     if(err == 1) {
<
Error(I18N(47, "Cannot create file %s"),fileName);
<                                     /* do
not update seek back info list for this file */
<                                     delFlag
= ERROR_DELETE;
<                                     break;
<                                     }
<                                     } else {
<                                     if(createFlag ==
DM_TRUE) {
<                                     if
(IS_FILE_SPARSE(fileCellp)) {
<
if (IS_DESTINATION_NTFS(fileName)) {
<
if (SET_FILE_TO_SPARSE(fd)) {
<
DebugPD(ASCII(" Inode no %d set to SPARSE successfully"
), fileCellp->inoNum);
---
>
if(err == 1) {

```

```

>
Error(I18N(47, "Cannot create file %s"),fileName);
>
/* do not update seek back info list for this file */
>
delFlag = ERROR_DELETE;
>
break;
>
}
>
} else {
>
if(createFlag == DM_TRUE) {
>
if (IS_FILE_SPARSE(fileCellp)) {
>
if (IS_DESTINATION_NTFS(fileName)) {
>
if (SET_FILE_TO_SPARSE(fd)) {
>
DebugPD(ASCII(" Inode no %d set to SPAR
SE successfully"), fileCellp->inoNum);
>
}
>
} else {
>
DebugPD(ASCII("Requested Sparse file %s not bei
ng set to sparse since detination FS is not NTFS"), fileName);
>
DebugPD(ASCII("Will attempt to restore the data
"));
>
}
463,465d484
<
} else {
<
DebugPD(ASCII("Requested Sparse file %s not being set to sparse
since detination FS is not NTFS"), fileName);
<
DebugPD(ASCII("Will attempt to restore the data"));
466a486,487
>
/* we don't require fd just now */
>
IGSclose64(fd);

```

```

468,471d488
<
<
require fd just now */
<
<
473,492c490,535
<
>size) == 0) {
<
keeping zero size files in file list */
<
AnnounceDone(fileName);
<
UX_ASSIGN(delFlag, FILE_DONE_DELETE);
<
} else {
<
back info and inode blk in rtrv data */
<
to make sure that this stream is data stream */
<
dataAttribCell =
(AttribStreamInfo *) GetFirstGenericList(attribListp);
<
mask = (dataAttribCell->type & ATTRIB_TYPE_MASK);
<
if
((NT_COMPARE(mask, MD_FILE_DATA_ATTRIB)) || (UX_COMPARE(mask,
MD_PRIMARY_ATTRIB))) {
<
int tmpSz = 0;
<
NT_ASSIGN(tmpSz, dataAttribCell->streamSize);
<
UX_ASSIGN(tmpSz, mdInode.dataSzToFollow);
<
NT_ASSIGN(fileData, ((char*)dataAttribCell->streamData));
<
UpdateRtrvDataTable(fileCellp,
<
tmpSz / sizeof (DataBlkCell),
<
(DataBlkCell *) fileData);
---
```

```

>
>                                     if
(GET_SIZE(inodeInfop->size) == 0) {
>
> /* no need of keeping zero size files in file list */
>
AnnounceDone(fileName);
>
UX_ASSIGN(delFlag, FILE_DONE_DELETE);
>                                     } else {
>
> /* insert seek back info and inode blk in rtrv data */
>
> /* table */
>
> /* here we have to make sure that this stream is data stream */
>
>
dataAttribCell = (AttribStreamInfo *)
GetFirstGenericList(attrbListp);

>
unsigned int mask = (dataAttribCell->type & ATTRIB_TYPE_MASK);
>
>
if ((NT_COMPARE(mask, MD_FILE_DATA_ATTRIB)) || (UX_COMPARE(mask,
MD_PRI
MARY_ATTRIB))) {
>
unsigned int tmpSz = 0;
>
NT_ASSIGN(tmpSz, dataAttribCell->streamSize);
>
UX_ASSIGN(tmpSz, mdInode.dataSzToFollow);
>
NT_ASSIGN(fileData, ((char*)dataAttribCell->streamData));
>
UpdateRtrvDataTable(fileCellp,
>
tmpSz / sizeof (DataBlkCell),
>
(DataBlkCell *) fileData);
>
}
>
>                                     break;
>
>
>                                     default:

```

```

>
DebugPD(ASCII("unknown file type for inode num : %u"),
>
fileCellp->inoNum);
>
>                                     delFlag =
ERROR_DELETE;
>
>                                     break;
>                                     } /* end switch -
handle files based on type */
>
>                                     switch (delFlag) {
>                                     case DO_NOT_DELETE:
>                                     /* keep files
nodes not to be deleted in regFileList */
>
AddToGenList(regFileListp, fileCellp);
>
>                                     break;
>                                     case FILE_DONE_DELETE:
>                                     if (createFlag
== DM_TRUE) {
>                                     /*
change attribs for files which are done now itself */
>                                     /* no
data is to be transfered from physical image */
>
mode_change(fileName,
>
inodeInfop->atime,
>
inodeInfop->mtime,
>
inodeInfop->uid,
>
inodeInfop->gid,
>
inodeInfop->mode);
494,519d536
<                                     }
<                                     break;
<
<                                     default:
<                                     DebugPD(ASCII("unknown file type
for inode num : %u"),
<
fileCellp->inoNum);
<
>                                     delFlag = ERROR_DELETE;
<                                     break;

```

```

<                                     }          /* end switch - handle files
based on type */
<
<             switch (delFlag) {
<                 case DO_NOT_DELETE:
<                     /* keep files nodes not to be
deleted in regFileList */
<                         AddToGenList(regFileListp,
fileCellp);
<
<                         break;
<                 case FILE_DONE_DELETE:
<                     if (createFlag == DM_TRUE) {
<                         /* change attribs for files which are done
now itself */
<
<                                     /* no data is to be
transferred from physical image */
<                                     mode_change(fileName,
<
<                                     inodeInfop->atime,
<
<                                     inodeInfop->mtime,
<
<                                     inodeInfop->uid,
<
<                                     inodeInfop->gid,
<
<                                     inodeInfop->mode);
<                                     }
521c538,544
<                                     Free(inodeInfop);
---
>
> Free(inodeInfop);
>
inodeInfop = NULL;
>
>                                     Free(fileCellp);
>
>
fileCellp = NULL;
>
>                                     break;
>                                     case ERROR_DELETE:
>
Free(inodeInfop);
523c546
<                                     Free(fileCellp);
---
>
>                                     Free(fileCellp);
525,531c548

```

```

<                                     break;
<
<                                     case ERROR_DELETE:
<                                         Free(inodeInfop);
<                                         inodeInfop = NULL;
<                                         Free(fileCellp);
<                                         fileCellp = NULL;
<                                         break;
---
>
>                                     break;
533,535c550,552
<
<                                     default:
<                                         break;
<
<                                     }
---
>
>                                     default:
>                                         break;
>
>                                     }
537,541c554,558
<                                     /*
<                                     * this function will take care of security and
named data. security
<                                     * will restored to a tmp file and named data
will be restored to actuall file
<                                     * this function is only for nt
<                                     */
---
>
>                                     /*
>                                     * this function will take care
of security and named data. security
>                                     * will restored to a tmp file
and named data will be restored to actuall file
>                                     * this function is only for nt
>                                     */
544,546c561,563
<                                     if(fileCellp != NULL) {
<
processNtFilePsiStream(fileCellp);
<                                     }
---
>
>                                     if(fileCellp != NULL) {
>
processNtFilePsiStream(fileCellp);
>                                     }
549,551c566,568
<                                     fileCellp = (FileInfoCell *)
GetNextGenericList(fileTable);
<                                     /* handle hard links if any */

```



```

<                while ((fileCellp != NULL) &&
(fileCellp->inoNum == mdInode.inoNum)) {
---
>                fileCellp = (FileInfoCell *)
GetNextGenericList(fileTable);
>                /* handle hard links if any */
>                while ((fileCellp !=
NULL) && (fileCellp->inoNum == mdInode.inoNum)) {
553,581c570,587
<                GetFileName(fileCellp->parentp,
GetNodeName(fileCellp),linkName);
<
<                DebugPD(ASCII(" link name is : %s "),
linkName);
<                if (createFlag == DM_TRUE) {
<                if (link(fileName, linkName) == -1) {
<                Log(I18N(97, "from : %s "),
linkName);
<                Log(I18N(98, "to : %s "), fileName);
<                Error(I18N(99, "Cannot create hard
link: %s"),
<                ErrorMsg(errno));
<                } else {
<                AnnounceDone(linkName);
<                }
<                }
<
<                Free(fileCellp);
<                fileCellp = (FileInfoCell *)
<                GetNextGenericList(fileTable);
<                }
<        }
<
<        tmpattribCell = NULL;
<        for ( tmpattribCell = (AttribStreamInfo *)
GetFirstGenericList(attribListp);
<                tmpattribCell != NULL;
<                tmpattribCell = (AttribStreamInfo *)
GetNextGenericList(attribListp)) {
<                if (tmpattribCell != NULL) {
<                if (tmpattribCell->streamData != NULL) {
<                Free(tmpattribCell->streamData);
<                Free(tmpattribCell);
---

```

```

>
GetFileName(fileCellp->parentp,
GetNodeName(fileCellp),linkName);
>
>
link name is : %s "), linkName);
>
DebugPD(ASCII("
if (createFlag
== DM_TRUE) {
if
(link(fileName, linkName) == -1) {
Log(I18N(97, "from : %s "), linkName);
>
Log(I18N(98, "to : %s "), fileName);
>
Error(I18N(99, "Cannot create hard link: %s"),
>
ErrorMsg(errno));
>
} else {
>
AnnounceDone(linkName);
>
}
>
Free(fileCellp);
fileCellp =
(FileInfoCell *)
>
GetNextGenericList(fileTable);
>
}
584,585d589
<
}
<
ResetGenList(attriListp);
587,590c591,602
< /* RKS+SALIL - should change this to reuse fileData */
< UX_FREE(fileData);
< fileData = NULL;
< }
/* end while - file metadata
processing */
---
>
tmpattribCell = NULL;
>
for ( tmpattribCell =
(AttribStreamInfo *) GetFirstGenericList(attriListp);
>
tmpattribCell != NULL;
>
tmpattribCell =
(AttribStreamInfo *) GetNextGenericList(attriListp)) {
>
if (tmpattribCell != NULL) {

```

```

>                                     if (tmpattribCell-
>streamData != NULL) {
>
Free(tmpattribCell->streamData);
>
Free(tmpattribCell);
>                                     }
>                                     }
>                                     ResetGenList(attribListp);
592c604,610
<     if (fileCellp != NULL)
---
>                                     /* RKS+SALIL - should change this to
reuse fileData */
>                                     UX_FREE(fileData);
>                                     fileData = NULL;
>                                     }/* end while - file metadata
processing */
>     }
>
>     if (fileCellp != NULL)
594,599c612,617
<         ErrorBegin();
<         Error(I18N(51, "Didn't get information about all files
in Metadata"));
<         Error(I18N(52, "Metadata looks to be incomplete"));
<         ErrorEnd();
<         retVal = DM_ERROR;
<         goto done;
---
>         ErrorBegin();
>         Error(I18N(51, "Didn't get information about all
files in Metadata"));
>         Error(I18N(52, "Metadata looks to be
incomplete"));
>         ErrorEnd();
>         retVal = DM_ERROR;
>         goto done;
603,606c621,624
<         DebugPD(ASCII("Cannot read file Metadata stream beyond
inode number : %u "),
<             currInoNum);
<         retVal = DM_ERROR;
<         goto done;
---

```

```
>             DebugPD(ASCII("Cannot read file Metadata stream
beyond inode number : %u "),
>                     currInoNum);
>             retVal = DM_ERROR;
>             goto done;
```

Exhibit M – Differences between versions 1.1.2.1 and 1.1.2.2 of rtrv_filemd.hpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.1 -r 1.1.2.2 rtrv_filemd.hpp
Index: rtrv_filemd.hpp
=====
===
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/rtrv_filemd.hpp,v
retrieving revision 1.1.2.1
retrieving revision 1.1.2.2
diff -r1.1.2.1 -r1.1.2.2
1c1
< /* $Id: rtrv_filemd.hpp,v 1.1.2.1 2001/02/10 09:41:26 nsq Exp
$ Copyright (c) 2001, Legato Systems, Inc. */
---
> /* $Id: rtrv_filemd.hpp,v 1.1.2.2 2002/10/19 22:31:12 nsq Exp
$ Copyright (c) 2002, Legato Systems, Inc. */
3,7d2
< /*
<  * Copyright (c) 2001, Legato Systems, Inc.
<  *
<  * All rights reserved.
<  */
20a16,17
> #include "dblklist.h"
>
27a25,58
> extern struct InodeIndexRec *inodeIndexTable;
> extern int fileMetadataFd;
> extern int newRestoreDesign;
>
> /* structs */
> typedef unsigned long CelestraOffset_t;
> typedef unsigned long CelestraCount_t;
> #if 0
> struct CelestraExtent {
>     short resv;                                /* unused currently, word
alignment */
>     unsigned short device;
>     CelestraOffset_t blockNumber;              /* ild = offset */
>     CelestraCount_t blockCount; /* ild = length */
> };
> typedef struct CelestraExtent CelestraExtent;
```

```

>
> struct CelestraExtentList {
>     int extentCount;
>     int extentAllocated;
>     int ildSize;
>     int ildAllocated;
>     unsigned char *ildData;
>     struct CelestraExtent *list;
> };
> typedef struct CelestraExtentList CelestraExtentList;
>
> struct MDFileBlockInfo {
>     long inoNum;                /* inode number: */
>     long offsetInFile;         /* starting offset of blocks
in file */
>     CelestraExtentList *blockList;    /* list of blocks */
> };
>
> extern struct MDFileBlockInfo MDFileBlockInfoList;    /*
file inodes info Table */
> #endif

```

Exhibit N – Differences between versions 1.1.2.10 and 1.1.2.11 of rtrvsinglepass.cpp

```
F:\codebase\si30\dev\igs\ndmpserver\modules\celestra\restorev2>x
:\cvs\cvs.exe diff -r 1.1.2.10 -r 1.1.2.11 rtrvsinglepass.cpp
Index: rtrvsinglepass.cpp
```

```
=====
---
RCS file:
/cvs/ipprod/cvs_root/dev/igs/ndmpserver/modules/celestra/restore
v2/Attic/rtrvsinglepass.cpp,v
retrieving revision 1.1.2.10
retrieving revision 1.1.2.11
diff -r1.1.2.10 -r1.1.2.11
2c2
< #ident "$Id: rtrvsinglepass.cpp,v 1.1.2.10 2001/09/20 10:32:57
harish Exp $ Copyright (c) 2001, Legato Systems, Inc."
---
> #ident "$Id: rtrvsinglepass.cpp,v 1.1.2.11 2002/10/19 22:33:05
nsq Exp $ Copyright (c) 2002, Legato Systems, Inc."
11c11
< static char rcsid[] = "@(#) $Id: rtrvsinglepass.cpp,v 1.1.2.10
2001/09/20 10:32:57 harish Exp $ " DM_BUILD;
---
> static char rcsid[] = "@(#) $Id: rtrvsinglepass.cpp,v 1.1.2.11
2002/10/19 22:33:05 nsq Exp $ " DM_BUILD;
17a18,20
> * Revision 1.1.2.11 2002/10/19 22:33:05 nsq
> * LGTpa45351: added code to use indexing of metadata to
perform FBF retrievals
> *
265a269
> #include "rip.hpp"
272a277,283
> /*
> * function prototypes
> */
> void* getChildInfoList(ino_t inode_num, __int64 offset,
DirChildInfoList *dirChildList, InodeCell **dirInfo);
> void newUpdateRetrievalTree(TreeNode *node);
> void UpdateDirInfo(TreeNode *currNode, InodeCell
*dirInfo);
>
431a443,451
> if ( newRestoreDesign ==1) {
> if (loadInodeIndexTable() < 0) {
```

1 of 3

```
432a453
> }
1147a1169,1226
> void newUpdateRetrievalTree (TreeNode *node)
> {
>     __int64 typeAndOffset;
>     DirChildInfoList *dirChildList = NULL;
>     DirChildInfoList *tmpDirChildList = NULL;
>     char *childName;
>     int bitOffset;
>     int byteOffset;
>     TreeNode *currChild;
>
>     // #ifndef FASTRAX
>     #if 0
>         if (ISUSEDINODE(node->inoNum) == DM_FALSE) {
>             DebugPD(ASCII("No need to process inode: %d"), node-
>inoNum);
>             return;
>         }
>     #endif
>
>     byteOffset = node->inoNum / NUMCHARBITS;
>     bitOffset = node->inoNum % NUMCHARBITS;
>     dirInodesMap[byteOffset] |= (1 << bitOffset);
>
>     typeAndOffset= inodeIndexTable[node->inoNum].typeAndOffset
> ;
>     DebugPD(ASCII(" in the newUpdateRetrievalTree : inodeNo :
%d, offset= %d"), node->inoNum, typeAndOffset);
>     if (typeAndOffset < (__int64)0) {
>         dirChildList = (DirChildInfoList *)getChildInfoList
(node->inoNum, typeAndOffset & (~DIR_BIT_MASK), dirChildList,
&node->inod
eInfo);
>         tmpDirChildList = dirChildList;
>         while(tmpDirChildList != NULL)
>     {
```

2 of 3

```

>         DebugPD(ASCII(" Child InodeName : %s is a child
returned by "), tmpDirChildList->fName);
>         tmpDirChildList = tmpDirChildList->nextElement;
>     }
> }
> currChild = node->link.childrenp;
> while(currChild != NULL)
> {
>     childName = (char *) GetNodeName(currChild);
>     DebugPD(ASCII(" ChildName : %s is a child of %s "),
childName, GetNodeName(node) );
>     tmpDirChildList = dirChildList;
>     while(tmpDirChildList != NULL)
>     {
>         if (strcmp(childName, tmpDirChildList->fName) ==
0)
>         {
>             currChild->inoNum = tmpDirChildList->inodNo;
>             typeAndOffset= inodeIndexTable[currChild-
>inoNum].typeAndOffset ;
>             if(typeAndOffset < (__int64)0)
>             {
>                 newUpdateRetrievalTree (currChild);
>             } else {
>                 currChild->inoNum = tmpDirChildList-
>inodNo;
>             }
>             break;
>         }
>         tmpDirChildList = tmpDirChildList->nextElement;
>     }
>     currChild = currChild->nextp;
> }
> }
1149a1229
>

```